DATE: A Framework for Supporting Design Artifact Tracking and Evolution

by

Christopher R. Vincent

Submitted to the Department of Electrical Engineering and Computer Science

in Partial Fulfillment of the Requirements for the Degrees of

Bachelor of Science in Electrical Engineering and Computer Science

and Master of Engineering in Electrical Engineering and Computer Science

at the Massachusetts Institute of Technology

May 21, 1999

[June 1999]

Author .....

Department of Electrical Engineering and Computer Science
May 21, 1999

Certified by .....

Howard E. Shrobe
Thesis Supervisor

Accepted by .....

Arthur C. Smith
Chairman, Department Committee on Graduate Theses

DATE: A Framework for Supporting Design Artifact Tracking and Evolution

by

Christopher R. Vincent

Submitted to the Department of Electrical Engineering and Computer Science

in Partial Fulfillment of the Requirements for the Degrees of

Bachelor of Science in Electrical Engineering and Computer Science

and Master of Engineering in Electrical Engineering and Computer Science

at the Massachusetts Institute of Technology

May 21, 1999

ABSTRACT

The DATE system supports collaborative engineering by providing a framework for capturing, archiving, evolving, and navigating design artifacts. For a software engineering effort, this includes source code, comments, test cases, documentation, and discussions between developers. Archives of design artifacts are rooted in a source code representation which addresses functional decomposition, configuration management, and version tracking. Users extend archives by operating on resources, asserting relationships (links) and attaching annotations such as documentation, observations, and design rationale. An initial implementation allows geographically distributed developers to navigate and extend a shared project archive using networked client interfaces. Supporting technologies are developed for leveraging object storage services (e.g. databases, file systems) and for efficiently sharing data across network and programming language boundaries.

Thesis Supervisor: Howard E. Shrobe

Title: Associate Director, MIT Artificial Intelligence Laboratory

Acknowledgements

Appreciation and thanks to Carolynn Bischoff, Andrew Blumberg, Sue Felshin, Roger Hurwitz, Robert Laddaga, John Mallery, Kalman Réti, Howard Shrobe, and Paul and Linda Vincent.

This paper describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the MIT Artificial Intelligence Laboratory's artificial intelligence research is provided in part by the Defense Advanced Research Projects Agency of the Department of Defense under contract number F30602-97-2-0013.

Contents

List of Figures

1 Overview

The DATE system supports collaborative engineering by providing a framework for capturing, archiving, evolving, and navigating design artifacts. For a software engineering effort, this includes source code, comments, test cases, documentation, and discussions between developers. Project archives provide an environment for creating and manipulating such design artifacts, allowing the integration of engineering support tools which have traditionally suffered from lack of a shared project representation.

A prototype DATE implementation, consisting of about 8,000 lines of Common Lisp and Java source code, provides an archive infrastructure targeting software engineering. A user interface demonstrates source code navigation, editing, and annotation capabilities made possible by the archive model. Supporting technologies are developed to enable system deployment over a distributed user base. Figure 1 illustrates the relationships between major DATE components, as outlined below.

Section 3: Archive Framework Archives, implemented in Common Lisp as networked servers, track the evolution of design artifacts generated by software engineering projects. At the core of the framework is a source code representation which addresses functional decomposition, configuration management, and version tracking. Users extend archives by operating on resources, asserting relationships (links) and attaching annotations such as documentation, observations, and rationale. The framework provides an intermediate level of representation, emphasizing support for design evolution and integration of tools. It offers far more power and flexibility than file-based software development archives, without incurring the overhead of full knowledge representation.

Section 4: User Interface A Java client allows geographically distributed groups of users to remotely navigate project archives, edit source code, and create resource annotations. The interface (Figure 5) demonstrates key features for a collaborative software evolution environment, strategically combining functionality from source code development, group discourse, and documentation tools.

Section 5: Persistent Object Support The archive is layered over a persistent object support service, which maps Common Lisp Object System (CLOS) classes to long-term storage mechanisms such as databases and file systems. The core application utilizes a simple transaction model which is insulated from details of how individual instances are stored, allowing it to act as an efficient

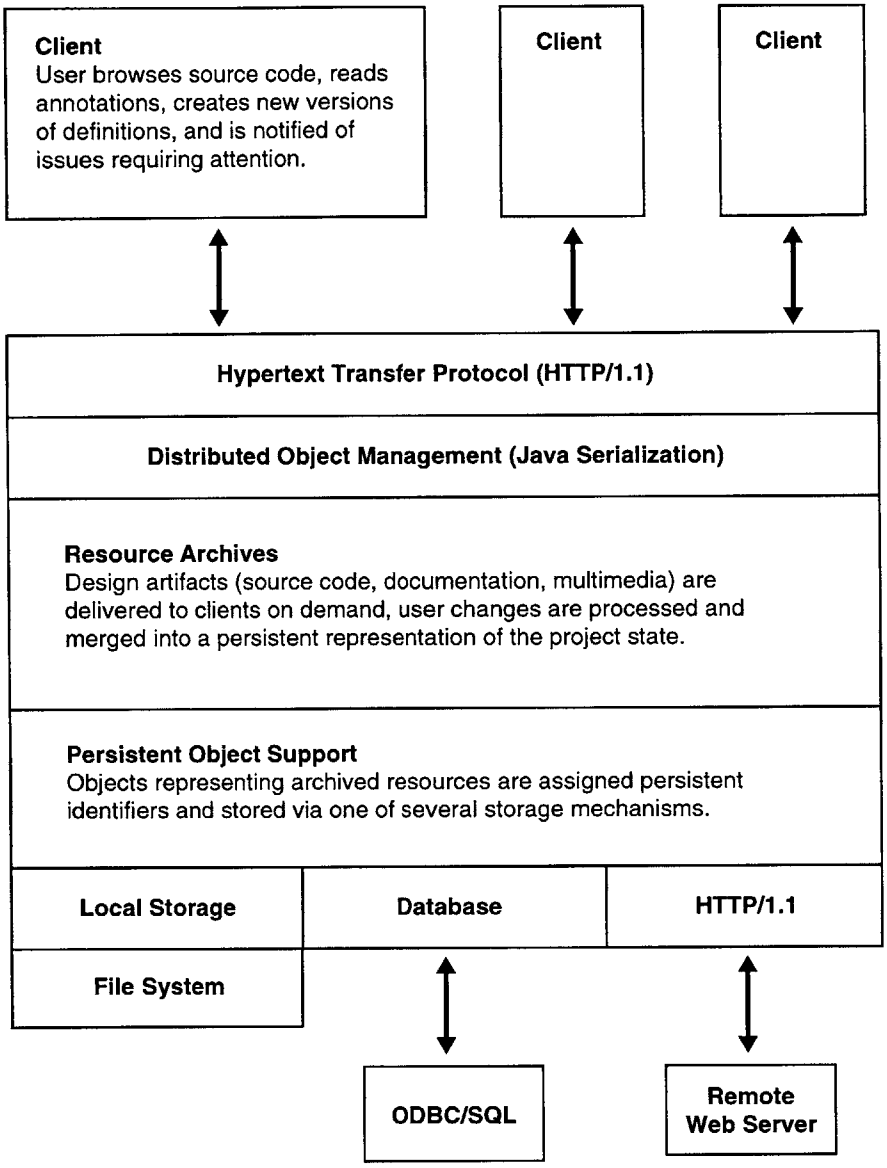


Figure 1: Overview of a collaborative software engineering support system.

clearinghouse for a heterogeneous set of resources.

Section 6: Distributed Object Management The archive server shares resources with clients using a remote object invocation service. It provides clients Lisp-to-Java translations of CLOS instances using the Java Object Serialization Specification. Programs may assert explicit control over the scope of data migration, allowing both servers and clients to optimize performance with caching and prefetch operations.

2 Design Artifacts

2.1 Artifacts Defined

The term *design artifact* is applied here rather broadly to denote all the information generated by a collaborative engineering effort. This includes not only traditional artifacts such as design diagrams and documentation, but observations, assertions, and design rationale that are seldom tracked effectively in contemporary engineering support environments. We should capitalize on the popularity of electronic communication (such as electronic mail) for collaborative discourse by maximizing its value to future design maintenance or reengineering efforts. While many such design artifacts are commonly stored (e.g. email archives), we need to explicitly represent the relationships between these resources as design artifacts in their own right. Simply capturing the link between an engineer's comment and the version of the design it describes adds a great deal of relevance to future users. The concept of design artifacts is applied to cases such as these, where the strategic application of existing design and communication technologies can be used to support more efficient, productive engineering teams.

2.2 Evolution of Designs

Some design artifacts, such as specifications, source code, and documentation tend to *evolve* as a sequence of discrete revisions. In a collaborative setting, these revisions often occur in response to observations, suggestions, or discussions by the members of an engineering team. While the change history of an artifact can be captured by version control systems with relative ease, the *rationale* leading to such changes, even if captured electronically, may be difficult to retrieve. A key observation is that an engineer may express the most valuable rationale once a design revision has already been completed. Explanations and clarifications occur frequently as part of normal collaborative discourse, and should be managed as valuable project assets. Evolving an existing system requires engineers to recreate the environment of design goals and trade-offs that led to a previous implementation. Artifacts supporting design evolution, whenever they are created, contribute to a design *genealogy*, outlining key points and issues that should not be neglected when later adapting a system.

3 Archive Framework

DATE archives represent a set of design artifacts and track their evolution as designs develop and adapt. Project resources and the relationships between them constitute nodes and links in a project web, which applications operate over in support of specific engineering tasks.

- **Representations target design evolution and tool integration.**
  DATE represents design artifacts at an intermediate level, establishing a level of granularity which applies across engineering support tools while providing a logical platform for building more powerful knowledge representations. Archives track the evolution of software engineering artifacts such as source code definitions and documentation.

- **Strategic application of domain analysis.**
  Archives integrate flexible graph constructs with uniform data structures derived from domain analysis. A source code representation based on functional decomposition, configuration management, and version tracking supplies a framework for software engineering archives.

3.1 Design Goals

The primary design goal of the DATE system is to facilitate powerful engineering support applications by managing the design artifacts related to a project. Such artifacts, or *resources*, include design rationale and diagrams, discourse between engineers, and in the case of software engineering, all the components of the product itself. We wish to cast the problem of system maintenance and adaptation as one of incrementally evolving original designs and design goals. To this end, we must maximize the value of design artifacts to future users, allowing engineers to adapt a system based on direct knowledge of previous design decisions and trade-offs. We wish to support a model of incremental adaptation, rather than one based solely on major milestones or system releases. In particular, care should be taken to preserve the minor changes that culminated in a release. Archives should continuously represent consistent project states, where consistency applies not to overall sys-

tem function, but to the integrity of individual resources and their relations. Users may still make inconsistent engineering changes, but we wish to support the timely discovery of such errors. Minimally, the DATE system should capture when and how modifications are made. At best, the system would support the automatic detection and containment of engineering conflicts.

Even a project involving only a handful of participants is expected to generate a large body of design artifacts. Re-engineering an existing system can become difficult or impossible when original decisions and rationale must be inferred due to lack of artifact acquisition or indexing. Engineers must assume the role of software archaeologists, searching for artifacts and theorizing as to their relationship to the task at hand. The convenience and availability of computers as a tool for design and discourse greatly ameliorates the resource acquisition problem, but there is little infrastructure available for archiving such resources in a useful manner. It is commonplace to produce electronic specifications and documentation, or to discuss an evolving design over electronic mail. These design artifacts are easily stored, but the relationships, or *links*, between resources are often ephemeral. It is these links, acting as persistent, bidirectional pointers, that give structure to an archive.

Resources and links combine to form a *web* of information concerning a collaborative project. We are especially interested in links that promote resource discovery in the course of normal engineering tasks (e.g. editing a source code definition). One important step in building useful archives is to capture links that, while immediately available at design time, tend to be lost as a system develops. A software engineering archive, for example, should link documentation or bug reports to the specific regions of source code they describe. Such program annotations are easily generated and stored, but offer a future developer little value unless linked to a particular state of the system's evolution.

A key trade-off in designing such a knowledge-based system is balancing domain analysis with archive flexibility. While we wish to minimize the amount of engineering methodology we impose on system users, domain analysis can be used to increase efficiency in representing and retrieving resources. In theory, we can represent any data structure as simple resources (nodes in a graph) and links, with a minimal number of link types. Links can accept other links as their predicates, further specifying a particular relationship. One powerful application of domain analysis is in defining a vocabulary of link types. This idea not only provides a more compact representation for links, it constrains the expression of engineering discourse such that automated, constraint-guided traversals of archives are more tractable. For example, links implying questions, agreement, disagreement, and alternatives are appropriate for a broad array of decision-making processes [7].

Domain analysis is also valuable in defining resource types. While we may support the archiving

of nearly any type of electronic data, certain regions of archive web will have inherently uniform structure. These cases can be isolated by domain, then represented and navigated much more efficiently. This type of domain knowledge can often be encoded without at all constraining a project's engineering methodology. Certain types of source code and documentation, for example, must rigidly conform to a predefined structure.

3.2 Resources

For the software engineering application of the DATE system, the domain analysis discussed above resulted in a data structure for representing source code. Since a software project naturally focuses on the program or programs being developed, this provides a consistent framework for all the project's associated design artifacts. Traditional artifacts such as design specifications, documentation, test cases, and bug reports branch off from the specific regions of source code they relate to. While a great deal of resource structure may be derived from source code stored in normal files, we wish to maintain a uniform representation that may be conveniently utilized by new applications. The following subsections outline key properties of the DATE source code representation, which is illustrated in Figure 2.

3.2.1 Granularity: Code Units

The representation should decompose source code into the smallest logical blocks that do not require deep understanding of programming language semantics. Here, we will define these blocks as *code units*. The selected decomposition should apply across various implementations and revisions of the programming language used. Note that while the DATE framework provides an appropriate platform for systems such as integrated development environments, detailed language semantics are not considered part of the core system. For many languages, the function definition is a natural example of an appropriate code unit.

3.2.2 Configuration Management: Implementations

The core source code representation should account for different *implementations* of the functionality delivered by a code unit. These implementations may apply to different hardware or software configurations, or to different engineering trade-offs. For example, it may be appropriate to maintain

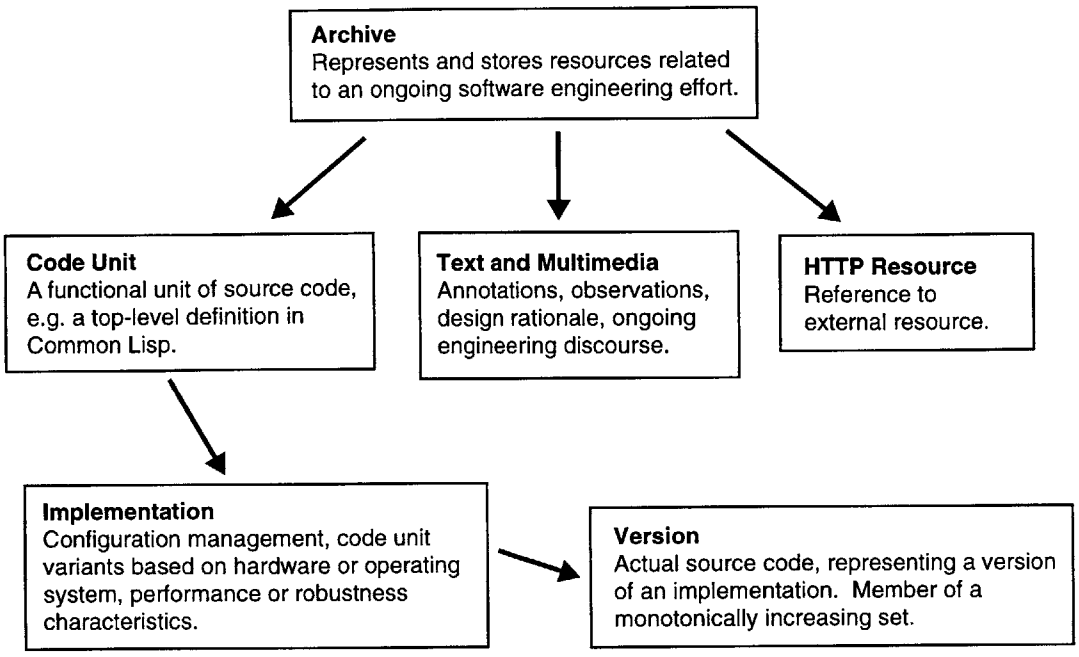


Figure 2: Code archive resource structure. Arrows represent a one-to-many relationship.

one implementation of a function that is highly optimized, while another captures debugging and metering information.

3.2.3 Representing Time: Versions

In order to maintain a coherent project history, the decomposition of source code functionality should be matched by an equally fine-grained representation of time. When an engineer links an annotation to a particular *version* of a source code definition, that link should not generally be broken. As changes are made, versions of an implementation are created as part of a monotonically increasing set. This feature is similar to the version control on files offered by many commercial source code management systems.

3.3 Application to Programming Languages

While the source code representation described in Section 3.2 may be applied to many programming languages, we selected Common Lisp for the demonstration system. The primary reason for this choice is the fine-grained abstractions allowed by the Common Lisp Object System (CLOS). Hinging on generic functions rather than data structures, CLOS allows methods to dispatch on a number

of classes, isolating common functionality as a first-class entity. In this case, code units are defined as top-level Lisp forms. These include definitions for generic functions, methods, classes, macros, and global variables. Implementations can vary by conventional Lisp compiler directives, or by the more qualitative, behavior-based aspects mentioned above. The Dylan programming language offers potential for fine-grained code unit manipulation similar to that of Common Lisp.

For more data-oriented languages such as Java and C++, we would decompose classes into their methods and instance variables. This is very useful for documentation and annotation purposes, but the prospects for fine-grained code unit reuse and automatic recomposition are weaker than in the CLOS model.

3.4 Links

While the source code representation described in Section 3.2 provides archives with a functional structure, it is the links between resources that drive collaborative engineering applications. Several categories of links are immediately applicable to the software engineering domain.

3.4.1 Language Semantics

These links encode relationships implicit in a system's source code, many of which are commonly utilized in software development environments. For example, such links connect methods to their generic functions, or function and variable references to their code units. The ability to create these links depends upon the level of support for programming language semantics. Some link types are easily derived, while others require a great deal of complexity (e.g. those depending on Common Lisp macro expansion).

3.4.2 Documentation and Specification

We assert that program-level specifications and documentation are much more valuable when persistently linked to the source code they are meant to describe. These important project components are traditionally encoded along with source code either as comments, or as part of a definition (as in Common Lisp documentation strings). Reifying specifications and documentation as archive resources means that they may be reasoned about and operated over at the same level as source code. Specifications can be associated with proofs or test cases, and users can make assertions about the validity of documentation. The problem of maintaining synchronization between source code and

documentation still persists, but is now contained in a rational manner. Documentation can be attached to any of the three code representation levels described in Section 3.2. Archives would serve as an excellent platform for a system which attempts to roll version-specific annotations forward as a definition is modified.

3.4.3 Annotation

Annotation is a rather broad term, used here to denote a link associating a resource with relevant information. The documentation and specification links described in Section 3.4.2 may be considered specific classes of annotation links. We are especially concerned, however, with the sort of loosely structured annotations, or programmer comments, often hampered by linear text editing technology. By separating the encoding of comments from that of source code, we remove restrictions on comment length, quantity, and media type which are inherent in their common representation. With the archive model in place, annotations are no longer restricted to plain text, taking the form of HTML, images, even audio and video.

Extensive comments have a way of hindering programmers who are familiar with a system, and of overwhelming novices. Maintaining comments as a separate, but closely linked counterpart to source code facilitates multiple program views, based on manual or machine-assisted selection of annotations. We also note the ability to maintain multiple annotation links to a single resource, as in a comment which applies to multiple regions of source code. The annotation not only describes the functional aspect that a region of code is part of, it acts as a navigational bridge to other, related components.

The ability to annotate previous annotations provides opportunities for collaborative discourse. The discussion threads common in mailing lists form within archives, with annotation trees branching off from the source code, documentation, or other resource that initiated the discourse. Electronic mail still plays an important role, but one of archive change notification rather than acting as the primary representation for design rationale. Traditional bug tracking email would take the form of notifications to those responsible for maintaining a resource. As discussed in Section 3.1, subclasses of annotation links are used to enforce particular models of discourse in discussion threads. This idea is especially useful for supporting the automatic traversal and characterization of annotation graphs.

3.4.4 Testing and Verification

The archive structure provides an excellent opportunity to closely integrate testing and verification information with source code, without sacrificing programmer convenience or program efficiency. A primary goal is to capture the ad-hoc test cases used by most programmers, but often discarded due to lack of a convenient archiving method. More formal, robust test sets may add value by facilitating automated regression testing. Test cases, depending on their scope and the state of program development, may be run whenever code units are modified. Adding fine-grained test cases (covering only one or a few code units) provides engineers with a convenient way to protect their assumptions against shifting abstractions or misinterpretation by future developers. Particular implementations of code units may be tested for overall performance, algorithm behavior, or even checked for consistency with a programmer-supplied proof.

3.5 Collections

While we assert that linear text is inadequate for representing source code and its associated design artifacts, files play an important organizational role in contemporary development environments. Editing a number of related functions in an Emacs [21] style buffer still offers a great deal of value, though we wish to add interface features for supporting archive navigation. Files are replaced by the concept of *collections*: linear, possibly overlapping sets of source code resources. Such collections may refer to specific versions or to code units in general, with a mechanism for automatically selecting which implementation and version to view. As with the selection of our source code representation, this concept is applicable to many programming languages. Collections yield their full advantage, however, when applied to languages built around fine-grained abstractions. Editing all the methods of a Common Lisp generic function, for example, may pull together code units from all over a large system.

We wish to accommodate the class-based, data-oriented source code view of Java and C++, as well as views based on one or more aspects of functionality. Both models are useful, and the programmer should not be forced to choose between them when storing implementations. Collections accommodate both constraint-based, dynamic views of source code, and manually assembled groups of resources meant to illustrate a particular idea. Collections may represent the current focus of an individual developer, or provide tutorials for new users. They may correspond to varying levels of abstraction, appropriate for tasks ranging from system orientation to low-level debugging.

3.6 Resource Indexing

While saved collections of resources provide an excellent basis for organizing and working with source code, we still require the ability to efficiently search an archive. Appropriate search constraints vary by archived data type, and by programming language for software engineering. It is not necessary that an archive efficiently map all artifacts, but almost all archives will need the ability to index some of their resources for quick retrieval later. Examples might include natural language queries based on documentation, or retrieval of function definitions by argument pattern. The demonstration archive performs indexing for code units (corresponding to Common Lisp definitions) based on their Lisp names. While top-level Lisp forms are not required to have unique names, definitions including those for functions, classes, and variables do. Apart from the normal archive web structure, we maintain lookup tables which allow the rapid retrieval of code units by name, package, and definition type.

4 User Interface

The user interface demonstrates several software engineering support features made possible by the DATE archive.

- **Dynamic source code views.**
  Files are replaced by logical groups of source code definitions, dynamically configurable to the task at hand.
- **Rationale capture by annotation.**
  Operating over a fine-grained source code representation adds new value to informal comments and discussions related to specific code.
- **Integration of tools.**
  Integration of tools for managing source code, documentation, and collaborative discourse promotes resource discovery and consistency.

4.1 Design Goals

The goal in designing the demonstration interface was not to implement a full-featured integrated development environment (IDE) or knowledge navigator. Rather, we wish to illustrate a set of software engineering support facilities which are implemented with relative ease when able to leverage the DATE archive framework. Accordingly, the demonstration screens shown are designed for intelligibility rather than compactness. The interface takes the form of a network client, supporting a great deal of local interaction, but closely tied to a remote project archive. The following subsections outline several features that the client interface was designed to demonstrate.

4.1.1 Fine-Grained Configuration Management

The configuration management provided by code unit implementations allows developers to compose coherent, task-oriented views from a large, heterogeneous base of source code. Most conventional source code representations force developers to make an unfortunate choice:

1. Separate platform-specific code from its logically similar, but portable counterparts.

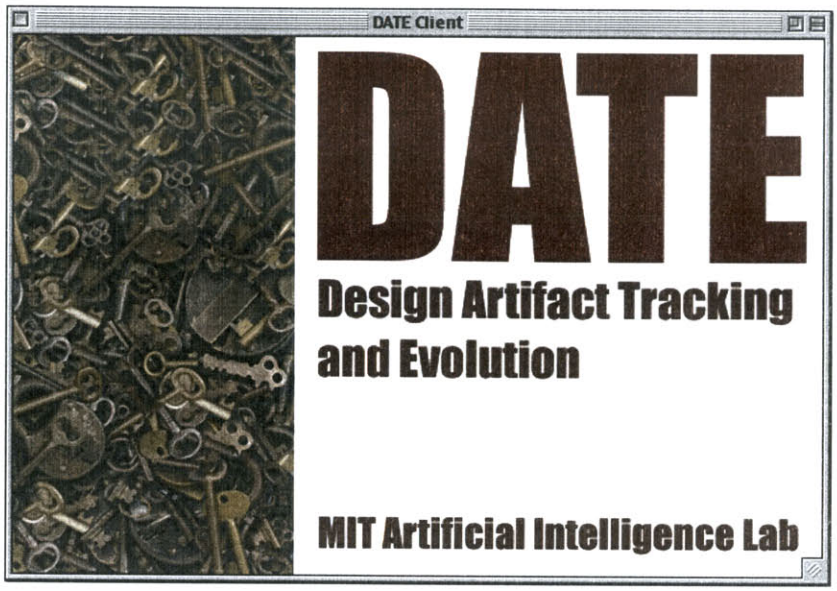


Figure 3: Client splash screen.

2. Accumulate multiple implementations in cluttered, highly conditionalized files.

With support for multiple, overlapping views, the developer may conveniently edit a buffer containing exactly the set of relevant definitions, regardless of selected configurations.

4.1.2 Multidimensional Source Code

The archive framework facilitates the development of multiple code views at two levels. First, the archive's decomposition of source code by functionality and time provides a rich space of resources. The demonstration interface allows users to view slices through this resource space, the collections discussed in Section 3.5. The configuration management discussed above is a specialized application of this idea. The archive framework is also appropriate for supporting lower-level multidimensional views of source code definitions. For example, a developer may wish to filter out type checking, debugging, or performance metering code from the current view. This recognizes the idea that a single definition in a contemporary programming language often corresponds to several semantic threads. The demonstation interface implements the first, higher level of multidimensional code support and provides a platform for developing the latter.

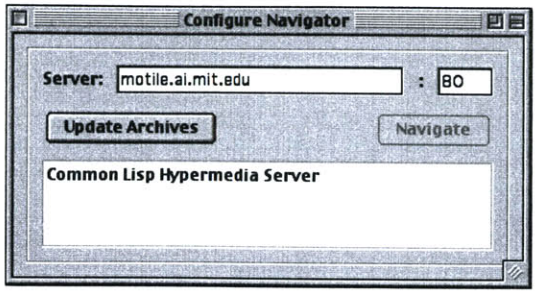


Figure 4: Client configuration window.

4.1.3 Integration of Tools

The archive representation provides a logical base for integrating source code development with tools for documentation, bug tracking, discourse, and project management. The demonstration interface utilizes the basic archive framework to support engineering discourse based on source code. When a user edits a source code definition, related design discussions, observations, and problems are immediately available for inspection.

4.2 Interface Components

The DATE demonstration interface is implemented as a Java application, communicating with archive servers via HTTP. Section 11.7 lists the source code for this application. The following subsections outline the main interface components.

4.2.1 Configuration

The client software first presents the user with a configuration window as shown in Figure 4. The user identifies the archive server of interest, selects `Update Archives`, and is presented with a list of available projects. The example in Figure 4 list only one archive, containing design artifacts for a web server written in Common Lisp. The user chooses the desired archive and selects `Navigate` to create an archive navigator window.

4.2.2 Archive Navigator

An archive navigator window, as shown in Figure 5, allows users to navigate, edit and create source code along with its associated design artifacts. The core feature of a navigator window is the editor

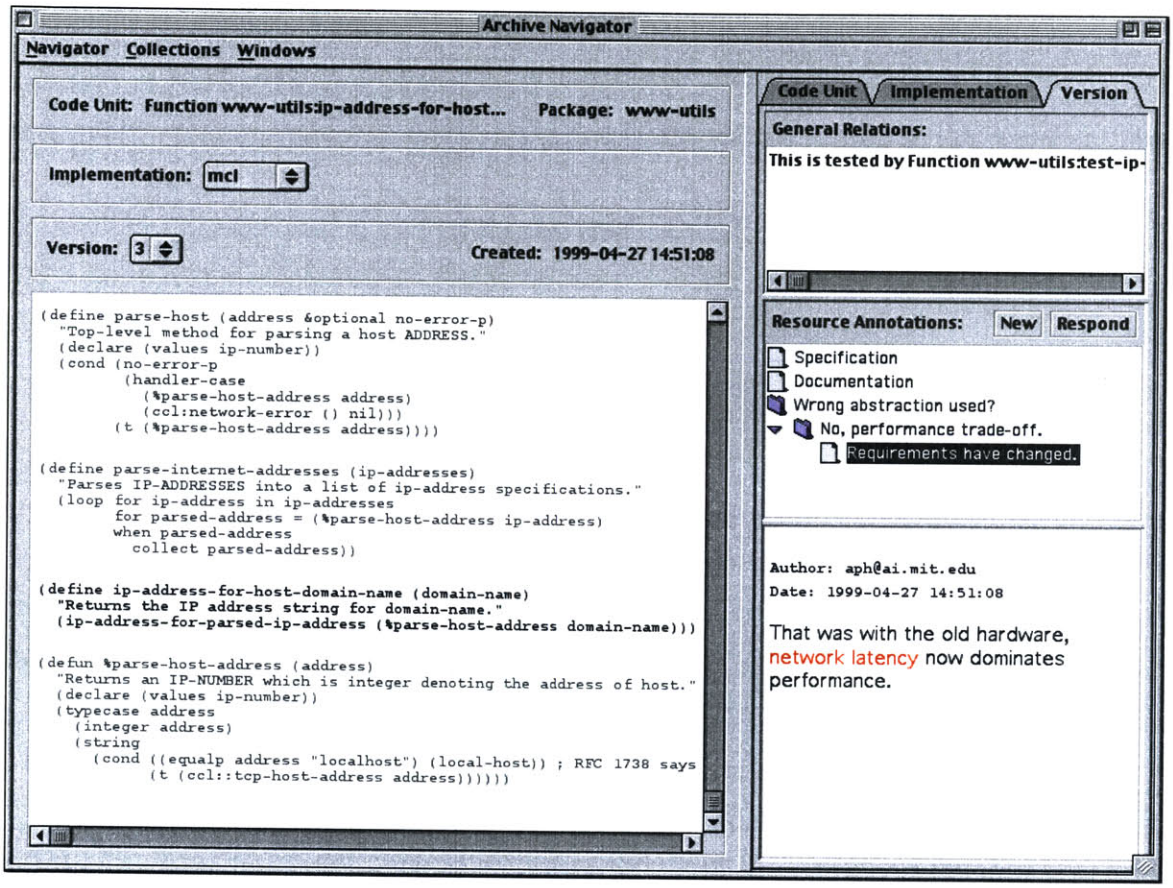
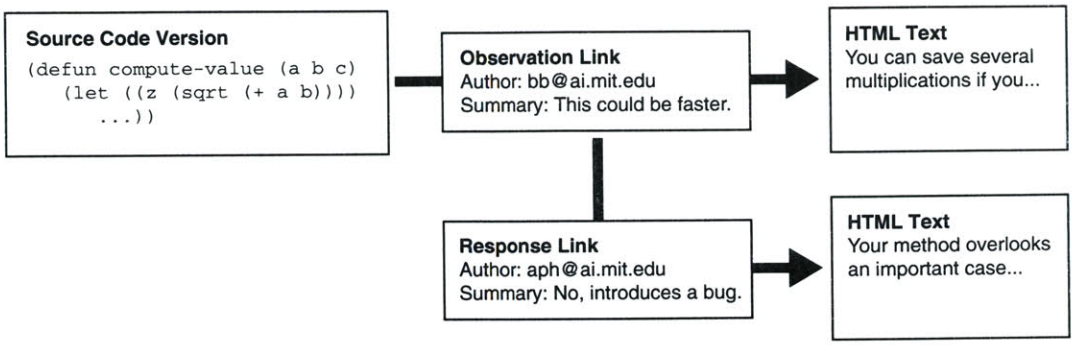
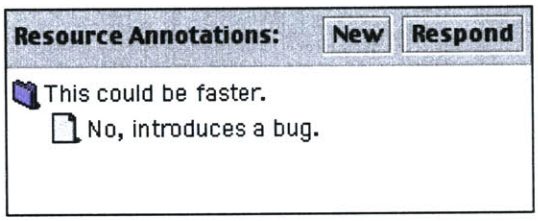


Figure 5: Archive navigator window.



(a)



(b)

Figure 6: (a) Archive and (b) interface representations of a discussion thread.

buffer, which displays versions of several code units simultaneously. The version currently being edited is considered the selected resource, and the rest of the window is dedicated to describing the local archive web. The resource information area directly above the editor buffer details the selected resource's code unit, implementation, and version. The user may select alternate implementations or versions, with such selections reflected in the editor buffer. The right-hand column displays links specific to one of the three code representation levels, as selected by the `Code Unit`, `Implementation`, and `Version` tabs. This allows, for example, a distinct separation of assertions on top-level code units from discussions concerning a particular implementation.

While all resource links may be rendered as English sentences, it is useful for the interface to provide special handling for selected link types. In the case of the demonstration system, annotation links are filtered out and displayed in the lower part of the link area. The client also searches for responses, represented in the archive as annotation links to other links of the same type (see Figure 6). In this manner, the interface presents a discussion tree, a simple example of which is shown in Figure 5. Selecting an annotation link from the tree displays the target resource in the lower-right

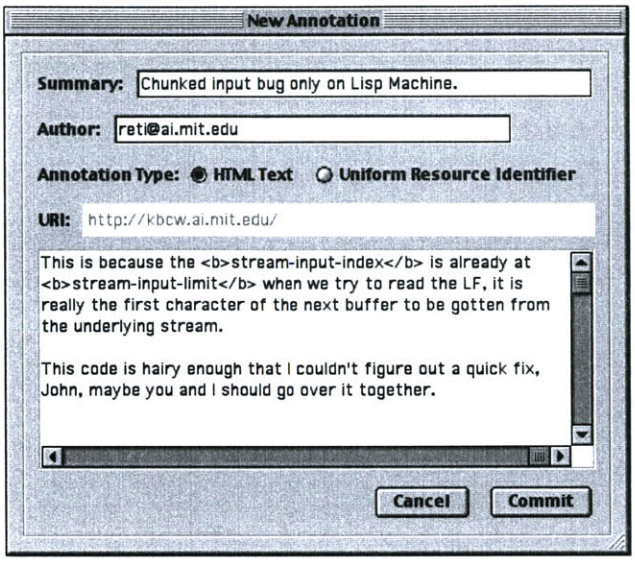


Figure 7: Annotation editor window.

area of the window, which provides display capabilities similar to those of a web browser. Other links, both from and to the selected resource, are displayed as sentences in the `General Relations` area. When such a link has a source code resource as its other predicate, selecting it appends that resource to the editor buffer.

4.2.3 Annotation Editor

Selecting `New` or `Respond` from the `Resource Annotations` area of the archive navigator displays the window shown in Figure 7. The annotation editor allows the user to specify a new annotation's summary, description, and author, and to create a new HTML text resource or external HTTP reference. If the editor was displayed by selecting `New`, the annotation link associates the selected source code resource with the new modifier resource. If `Respond` was selected, the HTML text or URI is associated with the currently selected link in the annotation tree. While the demonstration system only supports the creation of generic annotation links, the editor window is an appropriate interface for further quantifying engineering discourse. Users could select annotation link types such as bug report, improvement, agreement, disagreement, or alternative.

4.3 Editing Source Code

The demonstration client is built around a dynamic editor buffer, where source code is browsed and manipulated. While the example buffer shown in Figure 5 gives the impression of editing a source code file, it actually pulls together resources from a broad base of archived data. This is the logical approach for displaying the collections of source code discussed in Section 3.5. Users may start with saved collections of resources, then add and remove elements as they require additional information or shift focus. Several development environments for the Dylan programming language currently implement this kind of dynamic buffer functionality.

The navigator offers basic hypercode support (implemented as the *meta-dot* gesture familiar to Lisp programmers), adding requested code units to the dynamic buffer. For uniform, relatively compact resources such as source code definitions, this model of hypercode is more appropriate than the web browser style of "jumping" to a new resource. Rather than executing a one-dimensional walk of the archive's resource space, users vary the size of their working set, exploring the archive on many fronts. The currently selected resource is determined by the position of the insertion caret, and therefore may be changed using the mouse or arrow keys. In the example navigator window, the selected source code version (represented by its Lisp definition in the buffer) is colored black while all others are grey. While the demonstration system does not support saving new versions of definitions, it provides standard text editing features. We constrain normal editing such that the user may not select text which spans multiple definitions.

5 Persistent Object Support

Since the DATE system has at its core an archive of resources and links, efficient management of persistent storage is a crucial enabling technology. The design of the persistent object support layer focuses on a few key concepts:

- **Integration of storage mechanisms.**
  Objects are automatically distributed across appropriate storage mechanisms, which may include databases, file systems, or the web.
- **Persistent object identifiers.**
  Identifiers encapsulate references to remotely stored instances, acting as placeholders in partially loaded data structures.
- **CLOS programming style.**
  The transaction and class specification models are tailored to the Common Lisp Object System (CLOS) programming style. Slot accessors load stored instances transparently, and objects may be saved using method combination.

While this section presents persistent object technology for supporting Common Lisp, the key issues discussed apply across programming paradigms. Defining a generalized object persistence API both reduces source code complexity and insulates the compile-time system from volatile storage configuration issues. Such an interface must allow us to add support for emerging storage technolgies and to adjust storage mechanism selection criteria without any modifications to the core system.

5.1 Design Goals

5.1.1 Integration of Storage Mechanisms

We wish to insulate the archive system from direct database access, both to ease the transition to new storage methods and to provide a programming environment where persistence is mostly transparent. The ability to operate seamlessly over a heterogeneous group of *storage mechanisms* allows the archive to act efficiently as a clearinghouse for all the resource types related to a project.

Storage mechanisms might include databases, local or distributed file systems, and the World Wide Web, each offering performance characteristics appropriate for certain classes of data. One storage mechanism, for example, may be well suited to access patterns on source code while another is better equipped to store video.

5.1.2 Bounding Interaction

We require a persistent object layer that is well-suited to navigating a tangled web of objects, while remaining mostly transparent to the archive. We would like to be able to instantiate an object and operate on it normally without necessarily accessing the persistent state of all the objects bound to its slots. In this manner, well-contained archive traversals require a relatively small amount of interaction with storage mechanisms, and therefore less access to shared network resources. We assume that the most frequent archive operations will involve read-only walks of the archive web (e.g. browsing annotations, searching for source code), suggesting excellent prospects for object caching and prefetching.

5.1.3 Programming Interface

With the design constraints above in mind, there remains a great deal of flexibility in choosing the persistent object layer's programming interface. The two primary issues involved are choosing a type system and handling atomic modifications to the persistent state. The former is especially problematic for Common Lisp, since CLOS does not typically enforce constraints on slot types. This complicates, for example, mapping CLOS classes to a database schema which rigidly enforces field types. We initially experimented with imposing type specification on class slots, but eventually decided that this added too much of a burden for the programmer. In general, storage mechanisms are responsible for handling the loose typing traditionally supported in Lisp programming. Some storage mechanisms may require that the CLOS `:type` slot option is always used in the classes it stores in order to ease database schema mappings.

Supporting atomic changes to an object's (or to objects') persistent state is more a question of programming style. Two reasonable alternatives are to wrap object changes within a transaction macro (similar to `with-open-file`), or to explicitly save (snapshot) objects via a method call whenever appropriate. We chose the latter method primarily because it has less of a stylistic impact on the system's source code. Object saves may be hidden within CLOS `:after` methods, and easily

conditionalized. Any changes made to an object's slots are considered to be cached modifications. Saving one or a group of objects commits these modifications to the persistent state.

5.2 Storage Mechanisms

Storage mechanisms manage communication between the persistent object support interface and the actual modes of storage supported on the system. Slot access management and the assignment of persistent identifiers are handled in a centralized manner, so storage mechanism implementations tend to be quite compact. The demonstration system uses a portable log-based storage mechanism which utilizes the local file system. Section 11.3.7 lists the source code for this implementation, and Section 11.6.9 shows an example configuration. Earlier during system development, we used a platform-specific storage mechanism which translated to Statice, an object-oriented database for the Symbolics Lisp Machine (Section 11.3.16). The following subsections document the programming interface for allocating and manipulating storage mechanisms.

5.2.1 allocate-storage-mechanism

`allocate-storage-mechanism` *name class-name* `&key &allow-other-keys` *Generic Function*

Allocate an instance of a storage mechanism, keyword options vary according to *class-name*.

name	A unique symbol name to identify the instance.
class-name	Name of a subclass of `storage-mechanism`.

5.2.2 optimize-storage-mechanism

`optimize-storage-mechanism &key` *storage-mechanism verbose* *Function*

Calling this function allows a storage mechanism to perform computationally intensive time or space optimizations. Examples might include database compaction or file system defragmentation.

storage-mechanism	Storage mechanism to be optimized, defaults to `*default-storage-mechanism*`.
verbose	When non-`nil`, the storage mechanism may print status information to `*standard-output*`.

5.2.3 *default-storage-mechanism*

`*default-storage-mechanism*` *Parameter*

The default storage mechanism for saving instances of persistent classes.

5.2.4 *supported-storage-mechanisms*

`*supported-storage-mechanisms*` *Parameter*

A list of all the storage mechanisms currently in use.

5.3 Identifiers

An *identifier* contains all the information necessary to retrieve a persistent object. Identifiers consist of storage mechanism information, a unique serial number, and CLOS class information. An interesting property of identifiers concerns the class information, which enables an identifier to instantiate itself without accessing any persistent state. A skeleton instance, one with none of its slots filled, is created when an object is first referenced. Slot values are not fetched until the application first accesses persistent data. In this manner, an identifier may stand in for blind slot modifications (those not requiring other slot values) and allow those new values to be read or saved without reading any persistent state. Similarly, skeleton instances can be used for method dispatch before their persistent state has actually been retrieved.

An identifier is assigned to a persistent object the first time it is saved, and is associated with it throughout the object life-cycle. References to other persistent objects in the slots of that object are stored using their identifiers as proxies. The representations of an object across system restarts do not satisfy Lisp pointer equality (Lisp `eq`), but do represent equivalent application entities (Lisp `eql`). If two separate program threads request an object for the same identifier, they receive references to the same CLOS object. Therefore, as with normal CLOS objects, locking must be maintained at the application (in this case, archive) level.

5.4 Persistent Classes

The initial version of the persistent object layer utilized the Metaobject Protocol [14], using a metaclass to make the definition and use of persistent classes almost completely transparent. This approach proved not to be portable across Common Lisp implementations, so we moved to a variant of the `defclass` macro. This implementation constraint requires that persistent object slots are read and modified exclusively by their slot accessors, rather than `slot-value` or `with-slots`. The following subsections document the basic interface for defining and using persistent classes. The source code implementing this interface is listed in Section 11.3.12.

5.4.1 define-persistent-class

`define-persistent-class` *name* ({*superclass*}*) ({*slot-spec*}*) {*class-option*}* *Macro*

This wrapper macro for `defclass` adds support for persistent instances. Unless otherwise noted, arguments are the same as for `defclass`.

superclass — Classes defined by `define-persistent-class` automatically inherit from `persistent-instance-mixin`, which adds slots to hold an identifier and slot modification information.

slot-spec — The `:allocation` slot option may be specified as `:persistent`, denoting that a slot's value should be saved and restored as part of an instance's persistent state. If `:persistent` is not specified, a slot will be reset to to its `:initform` value or left unbound after a system restart. Some storage mechanisms may require the `:type` slot option in order to store instances of a persistent class. Slot specifications inherit all slot options except `:allocation`.

5.4.2 make-persistent-instance

`make-persistent-instance` *class* `&rest` *initargs* *Function*

This variant of `make-instance` creates a new instance of a persistent class, which may later be saved using `save-object`. Slot values must be accessed using slot accessors, rather than `slot-value` or `with-slots`. Unless otherwise noted, arguments are the same as for `make-instance`.

class — The class specified must have been defined using `define-persistent-class`.

5.4.3 save-object

`save-object` *instance* `&key` *storage-mechanism if-exists* *Generic Function*

Saves an object created by `make-persistent-instance`, assigning it a new persistent identifier if previously unsaved.

instance — An instance to be saved, created using `make-persistent-instance`.

storage-mechanism — Storage mechanism used for saving *instance*, defaults to `*default-storage-mechanism*`.

if-exists — Controls behavior when *instance* has been previously saved. Possible values are `:overwrite` (default) and `:error`.

5.4.4 save-objects

`save-objects` *instance-list* `&key` *storage-mechanism if-exists* *Generic Function*

Applies `save-object` atomically to a list of instances. All unsaved instances are assigned identifiers before any instances are saved. Unless otherwise noted, arguments are the same as for `save-object`.

instance-list — A list of instances to be saved, each created using `make-persistent-instance`.

storage-mechanism — Storage mechanism used to save previously unsaved instances, defaults to `*default-storage-mechanism*`.

5.4.5 object-saved-p

`object-saved-p` *instance* *Generic Function*

Returns non-`nil` if *instance* has been previously saved using `save-object` or `save-objects`.

instance	An instance created using `make-persistent-instance`.

5.4.6 object-modified-p

`object-modified-p` *instance* *Generic Function*

Returns non-`nil` if *instance* has been modified since it was created or last saved.

instance	An instance created using `make-persistent-instance`.

5.4.7 object-cached-p

`object-cached-p` *instance* *Generic Function*

Returns non-`nil` if the slot values for *instance* are cached, and thus can be read without accessing its storage mechanism. Applications usually do not need to call this method directly.

instance	An instance created using `make-persistent-instance`.

5.4.8 object-identifier

`object-identifier` *instance* *Generic Function*

Returns the persistent identifier for *instance*. Causes an error if the instance has not yet been saved using `save-object` or `save-objects`. Applications usually do not need to call this method directly. Direct access to identifiers can be useful for building efficient network interfaces and indexing of objects.

instance	An instance created using `make-persistent-instance`.

5.4.9 load-object

`load-object` *identifier* `&key` *force-cached-p* *Generic Function*

Returns an instance for a persistent identifier. If another thread is using the instance associated with *identifier*, the same instance is returned. If the program has previously referenced and released such an instance, that instance may be returned. Applications usually do not need to call this method directly.

identifier	A persistent identifier.
force-cached-p	If non-`nil`, the instance returned is guaranteed to have its slots already cached (retrieved from persistent storage and bound as slot values).

5.4.10 save-root-object

`save-root-object` *thing* `&key` *storage-mechanism* *Generic Function*

Saves a persistent instance, persistent identifier, or primitive Lisp type as the root object for a storage mechanism. This is useful for boot-strapping persistent data structures, e.g. by saving a hash table which references a number of persistent instances.

thing	A persistent instance, persistent identifier, or primitive Lisp type.
storage-mechanism	Storage mechanism used to save *thing*, defaults to `*default-storage-mechanism*`.

5.4.11 load-root-object

`load-root-object` `&key` *storage-mechanism* *Generic Function*

Returns a root object saved using `save-root-object`.

storage-mechanism	Storage mechanism containing the desired root object, defaults to `*default-storage-mechanism*`.

5.5 Example

The following example illustrates how to define a persistent class, create new persistent instances, and save those instances to the default storage mechanism. First, we define the persistent class and variables used in the example.

```
(define-persistent-class my-persistent-class ()
  ((first-slot
    :allocation :persistent
    :initarg :first-slot
    :accessor get-first-slot)
   (second-slot
    :allocation :persistent
    :initarg :second-slot
    :accessor get-second-slot))
  (:documentation "A demonstration persistent class."))

(defvar *object-1*)

(defvar *object-2*)
```

With the persistent class defined, we may create instances using `make-persistent-instance`.

```
(setf *object-2*
      (make-persistent-instance 'my-persistent-class
                                :first-slot 123
                                :second-slot nil)
      *object-1*
      (make-persistent-instance 'my-persistent-class
                                :first-slot :test
                                :second-slot *object-2*))
```

Now, we use the persistent object layer's introspective capabilities to examine an instance before and after the objects are saved.

```
(object-saved-p *object-1*)   →   nil
(object-modified-p *object-1*)   →   t
(save-objects (list *object-1* *object-2*))
(object-saved-p *object-1*)   →   t
(object-modified-p *object-1*)   →   nil
```

We save a root object before restarting the system. Applications would typically use a hash table or vector as a root object, supplying references to a number of persistent instances.

```
(save-root-object *object-1*)
<SYSTEM RESTART>
```

Following a system restart, we use the root object for the default storage mechanism as a reference to our persistent data structure.

```
(setf *object-1* (load-root-object))
```

With `*object-1*` bound, we use `object-cached-p` to illustrate the automatic retrieval of slot values from persistent storage.

```
(object-cached-p *object-1*)   →   nil
(get-first-slot *object-1*)   →   :test
(object-cached-p *object-1*)   →   t
```

When the slot values for `*object-1*` were cached, a skeleton instance was created for the object bound to `second-slot`. We now retrieve this instance, read one of its slots, then make a modification to another slot.

```
(setf *object-2* (get-second-slot *object-1*))
(object-cached-p *object-2*)   →   nil
(get-first-slot *object-2*)   →   123
(object-cached-p *object-2*)   →   t
(object-modified-p *object-2*)   →   nil
```

```
(setf (get-second-slot *object-2*) *object-1*)
(object-modified-p *object-2*)   →   t
```

6 Distributed Object Management

The DATE server distributes regions of archive web to its clients, translating Common Lisp values to serialized Java objects in their canonical binary form. Java-to-Lisp remote method invocation (RMI) extends this model by providing direct access to server-side computation and archive traversal. Key features include:

- **Portable, efficient data translation.**
  Portable Common Lisp code translates values directly to the binary format defined in the Java Object Serialization Specification.
- **Highly adaptable distribution policies.**
  Servers assert explicit control over the scope of data duplication and distribution. Requests for single resources may be serviced with larger regions of archive web, as constrained by prototypical access patterns, client prefetch requests, and server load.

While this section addresses issues specific to communication between Common Lisp and Java, the discussion is applicable to general problems in data sharing across network and programming language boundaries. We concentrate on developing an interface that automates object translation while providing support for aggressive, dynamic performance optimizations.

6.1 Design Goals

We assert that in a data-centric, distributed application such as the DATE framework, explicit, programmable control over data migration should be a primary system function. While distributed object systems such as CORBA can be effective in navigating webs of objects, we wish to further expose the mechanisms for caching and prefetching regions of object graphs. In a networking paradigm where latency is a key performance inhibitor, both clients and servers should be given a good deal of flexibility in reducing the frequency of round-trip requests. When determining the scope of data prefetching, servers must have the ability to consider infrastructure-specific constraints (network load, capacity) and application-specific observations (common graphs walks, client feedback) in the

same logical computation. At the same time, clients should have the ability to probabilistically refine prefetching and caching based on algorithm characteristics.

In order to support the evolution of such capabilities, we allow the archive server to explicitly pass regions of resource web to its clients. Consumers of such data may operate over it using shared abstractions, or potentially through automatically generated, portable interfaces. Rather than conforming to the method-based object interface of CORBA, client-side traversal algorithms may weigh the value of prefetched graph regions against that of remote data which is not immediately available. In addition, explicit knowledge of data migration should assist servers and clients in dynamically altering rules for selecting distributed versus server-side computation.

We are initially interested in allowing Java-based clients to interact with an archive server developed in Common Lisp. The Java Object Serialization Specification [10] defines stream-based, binary representations for objects and primitive types appropriate for this task. When applied in conjunction with the persistent object layer discussed in Section 5, we can assert fine-grained control over the copying and distribution of regions of archive web. The client component of the demonstration system takes advantage of these capabilities by batching prefetch requests for resources it anticipates requiring.

6.2 Remote Method Invocation

Distributed object management in the DATE system takes place primarily through Java to Lisp remote method invocation (RMI). The network client discussed in Section 4 uses this model of communication exclusively in pulling data from the archive server. In most cases, the client directly accesses the core archive interface to request and manipulate resources. This exposes abstractions to a wider base of software, but facilitates rapid development by eliminating the need for a specialized network protocol.

In its simplest form, the Lisp RMI interface gives privileged Java clients direct access to `funcall`. Leveraging HTTP/1.1 [4] as a transport layer, the interface accepts a function name and list of arguments, performs the requested computation, and returns the result. While HTTP is often thought of in conjunction with web browsers, the specification has actually been engineered as a generic transport layer. It is tailored to relatively small resources, with optimizations favoring the duplication and distribution of resources over upstream client modifications. This model is well-suited to the application at hand, where we expect users to observe many more resources than they

Lisp	Java
`nil`	`null`
`t` `nil`	java.lang.Boolean
integer	java.lang.Byte java.lang.Short java.lang.Integer java.lang.Long
float/rational	java.lang.Float java.lang.Double
character	java.lang.Character
string	java.lang.String
vector	array(java.lang.Object)
cons	java.util.Vector
hash-table	java.util.Hashtable
symbol	lisp.lang.Symbol
keyword	lisp.lang.Keyword
instance	lisp.map.*

Figure 8: Lisp-Java type mapping used for Java object serialization.

create. Utilizing HTTP rather than simple TCP/IP connections allows us to take advantage of an existing caching, compression, and security infrastructure. We may assert additional programmatic control over the caching and distribution of results, operating on arguments and return values in their canonical binary form.

6.3 Lisp Interface

The following subsections document the Lisp interface for communicating with Java object serialization streams. Lisp programs may automatically read and write primitive types, subject to the translations listed in Figure 8. Serializing instances of CLOS classes requires no additional Lisp code, but a new Java class file must be generated before Java programs can deserialize a particular class. These class files could potentially be generated and distributed on demand, completely automating the serialization of new or redefined CLOS classes. The source code implementing this interface is listed in Section 11.2.2.

6.3.1 with-java-serialization-output

`with-java-serialization-output` (*stream*) `&body body` *Macro*

Creates an environment in which Lisp values may be serialized to *stream* using `write-object`.

stream	A binary output stream.

6.3.2 write-object

`write-object` *thing stream* *Function*

Writes *thing* as a serialized Java object to *stream*, within the `with-java-serialization-output` environment.

thing	Any CLOS instance or Lisp primitive type supported by the Java serialization interface.
stream	The binary output stream passed to `with-java-serialization-output`.

6.3.3 output-byte-count

`output-byte-count` *Function*

Returns the number of serialization output bytes written within the `with-java-serialization-output` environment.

6.3.4 with-java-serialization-input

`with-java-serialization-input` (*stream*) `&body body` *Macro*

Creates an environment in which Java objects may be deserialized as Lisp values from *stream* using `read-object`.

stream	A binary input stream.

6.3.5 read-object

`read-object` *stream* `&key` *eof-errorp eof-value* *Function*

Reads a serialized Java object from *stream* as a Lisp value, within the `with-java-serialization-input` environment.

stream	The binary input stream passed to `with-java-serialization-input`.
eof-errorp	If supplied and non-`nil`, an error is signaled when the end of input is reached.
eof-value	If *eof-errorp* is `nil`, this value is returned when the end of input is reached. Defaults to `nil`.

6.3.6 input-byte-count

`input-byte-count` *Function*

Returns the number of serialization input bytes read within the `with-java-serialization-input` environment.

6.3.7 instance-for-serialization

`instance-for-serialization` *instance depth* *Generic Function*

Applications may modify or replace instances before they are translated to their Java representations. This is useful, for example, in limiting the extent of data structure serialization.

instance	An instance passed to `write-object` either directly or in the course of serializing another Lisp value.
depth	The number of nested calls to `write-object` that resulted in this call.

6.3.8 generate-java-source-for-class

`generate-java-source-for-class` *class-name directory* *Function*

Writes a Java source file mapping the CLOS class *class-name* to it's Java representation. Returns the pathname of the new source file, which must be compiled before its class is referenced in Java.

class-name	The name of the CLOS class mapped to Java.
directory	Pathname for the directory where the Java source file is created.

6.3.9 flush-cached-class-mappings

`flush-cached-class-mappings` *Function*

Flushes cached class mappings, should be called after serializable CLOS classes are modified.

6.4 Java Interface

Since all serialized Lisp values are mapped to subclasses of `java.lang.Object`, the only serialization methods appropriate for communication with Lisp are `readObject` on `java.io.ObjectInputStream` and `writeObject` on `java.io.ObjectOutputStream`. Therefore numbers, characters, and boolean values must be serialized within their appropriate wrapper classes. Primitive Lisp types that map to existing Java classes operate seamlessly with Java applications, not requiring that any additional packages or classes be available. Lisp types for which an appropriate, predefined mapping does not exist are defined in the `lisp.lang` package. The only mappings currently implemented in this manner are `lisp.lang.Symbol` (Section 11.7.14) and `lisp.lang.Keyword` (Section 11.7.12).

Accessing serialized representations of CLOS instances requires some additional support. Before a CLOS class may referenced in Java, a corresponding Java class must be created in the `lisp.map` package. The CLOS inheritance structure of a class is flattened such that all accessible Lisp slots become direct Java instance variables of type `java.lang.Object`. Once this mapping is available, a deserialized object may be cast to its class and its instance variables accessed directly. For programmer convenience, all Lisp class mappings inherit from `lisp.lang.LispObject`, providing runtime variable access through the `getSlotValue` method. Using this model of type translation to traverse a data structure requires approximately one Java cast for each implicit type assumption in a corresponding Lisp program.

6.5 Examples

6.5.1 Primitive Lisp Types

As discussed above, serializing primitive Lisp types does not require any special action by the Lisp or Java programmer. For illustration purposes, we write the serialized Java data to a binary file. In an application, values are usually serialized directly to a binary network connection. The following Lisp expression serializes several values using `write-object`.

```
(with-open-file (stream (pathname "test-output")
                        :direction :output
                        :if-exists :supersede
                        :if-does-not-exist :create
                        :element-type '(unsigned-byte 8))
  (with-java-serialization-output (stream)
    (write-object "This is a string." stream)
    (write-object 123456789 stream)
    (write-object #\x stream)
    (write-object (make-array 3 :initial-contents '(1 "Two" 3)) stream)))
```

Now that the serialized values are stored in the file named `test-output`, we may execute the following Java code to read the values as Java objects.

```
FileInputStream file_stream = new FileInputStream("test-output");
ObjectInputStream obj_stream = new ObjectInputStream(file_stream);
for (int i = 0; i < 3; i++) {
  Object obj = obj_stream.readObject();
  System.out.println(obj.getClass().getName() + ": " + obj);
}
Object[] array = (Object[])obj_stream.readObject();
for (int i = 0; i < array.length; i++) {
  System.out.println(array[i].getClass().getName() + ": " + array[i]);
}
```

```
System.out:

java.lang.String: This is a string.
java.lang.Integer: 123456789
java.lang.Character: x
java.lang.Byte: 1
java.lang.String: Two
java.lang.Byte: 3
```

The following Lisp code illustrates deserializing the same Java objects using `read-object`.

```
(with-open-file (stream (pathname "test-output")
                        :direction :input
                        :element-type '(unsigned-byte 8))
  (with-java-serialization-input (stream)
    (loop for thing = (read-object stream :eof-value 'eof)
          until (eq thing 'eof)
          collect thing)))

→   ("This is a string." 123456789 #\x #(1 "Two" 3))
```

6.5.2 CLOS Objects

We start by defining a CLOS class and generating a source file for the corresponding Java class. In this example, `*java-mappings-pathname*` denotes the directory where classes in the package `lisp.map` are stored.

```
(defclass test-class ()
  ((first-slot
    :initarg :first-slot)
   (second-slot
    :initarg :second-slot))
  (:documentation "A demonstration serializable class."))

(generate-java-source-for-class 'test-class *java-mappings-pathname*)
```

We defined `test-class` in the `:jos` package, so the Java representation of the class is named `Jos_TestClass`. Calling `generate-java-source-for-class` generated the following Java source code, which must now be compiled.

```
package lisp.map;

import java.io.Serializable;
import lisp.lang.*;

public class Jos_TestClass extends LispObject implements Serializable {

private static final long serialVersionUID = 1;

public Object firstSlot;
public Object secondSlot;
```

```
public Jos_TestClass () {}

}
```

We may now create an instance of the CLOS class `test-class` and serialize it as an instance of the Java class `Jos_TestClass`, again using the file named `test-output`. Note that we serialize a Lisp keyword, which requires the Java support class `lisp.lang.Keyword`.

```
(with-open-file (stream (pathname "test-output")
                        :direction :output
                        :if-exists :supersede
                        :if-does-not-exist :create
                        :element-type '(unsigned-byte 8))
  (let ((instance (make-instance 'test-class
                    :first-slot "This is a string."
                    :second-slot :test)))
    (with-java-serialization-output (stream)
      (write-object instance stream))))
```

After including `lisp.lang.*` and `lisp.map.Jos_TestClass`, the following code reads the Java representation of the CLOS instance generated above. Note that the same instance variables may be accessed in two distinct ways. First, we cast the deserialized object to `Jos_TestClass` and access its variables directly. Second, we cast the object to a generic `LispObject`, and perform run-time variable access.

```
FileInputStream file_stream = new FileInputStream("test-output");
ObjectInputStream obj_stream = new ObjectInputStream(file_stream);
Object obj = obj_stream.readObject();
Jos_TestClass direct_obj = (Jos_TestClass)obj;
System.out.println(" firstSlot: " + direct_obj.firstSlot);
System.out.println("secondSlot: " + direct_obj.secondSlot);
LispObject lisp_obj = (LispObject)obj;
System.out.println(" firstSlot: " + lisp_obj.getSlotValue("firstSlot"));
System.out.println("secondSlot: " + lisp_obj.getSlotValue("secondSlot"));

System.out:

 firstSlot: This is a string.
secondSlot: :TEST
 firstSlot: This is a string.
secondSlot: :TEST
```

Finally, we may also deserialize the instance in Lisp using `read-object`.

```
(with-open-file (stream (pathname "test-output")
                        :direction :input
                        :element-type '(unsigned-byte 8))
  (with-java-serialization-input (stream)
    (with-slots (first-slot second-slot) (read-object stream)
      (values first-slot second-slot))))

→   "This is a string." :TEST
```

7 Evaluation

For demonstration and evaluation purposes, the DATE framework was applied to source code for the Common Lisp Hypermedia Server (CL-HTTP) [18]. The system implements full-featured HTTP server, client, and proxy services on eight major Lisp platforms. The project has benefited from five years of distributed software engineering collaboration, evolving incrementally from 2,700 lines of Common Lisp code to over 50,000 lines and more than 7,300 top-level definitions.

7.1 Archive Framework

The evaluation archive contains all CL-HTTP sources from the Macintosh Common Lisp (MCL) and Symbolics Lisp Machine (Genera) configurations. The code listed in Section 11.8 walks the system's source tree, pulling out top-level Lisp definitions and installing them in the archive. This provides us with a substantial base of source code resources to operate over. The CL-HTTP system is designed such that a good deal of configuration management takes place at the definition level, e.g. a function might be defined once for each platform. Over 300 such top-level conditionalizations apply to definitions in the MCL and Genera ports, represented in the archive as code units with more than one implementation. While more qualitative implementation variants (e.g. performance/safety trade-offs) are difficult to detect automatically, these platform-based variants are sufficient for demonstrating the archive's source code structure.

7.2 Persistent and Distributed Object Support

The evaluation archive was large enough to detect and address performance bottlenecks in the implementations of persistent object support (Section 11.3) and distributed object management (Section 11.2). The archive utilizes a single, portable storage mechanism (Section 11.3.7) which stores objects in the local file system. Without implementing a full-featured database, we were able to produce a self-contained archive that comfortably scales to accommodate CL-HTTP.

While the demonstration system is targeted for a single host, the client still communicates with the server via HTTP. This allows us to evaluate and refine the distributed object management technology discussed in Section 6. Shortcomings in Java's HTTP implementation, along with running the DATE server and client on the same physical host, exaggerate the cost of network communication and emphasize the performance characteristics of resource prefetching. Even simple client and server-

side prefetching policies delivered measurable improvements in interface responsiveness.

8 Future Work

8.1 Building a Navigation Toolkit

Implementing and evaluating the demonstration system highlighted the need for a mechanism to describe common link structures in the archive web. We initially combine two concurrent models of web structure. The first, as in our source code representation, allows highly functional, but rigid data structures derived from domain analysis. The second supports the evolution of highly flexible, though possibly anarchic regions of free-form link structure. There should be an intermediate structural representation that spans the gap between these two extremes, capturing and standardizing common patterns in local webs. Promoting these patterns to first-class entities would make automated traversal of common design structures more tractable, without eliminating the evolution of new patterns through the addition of loosely constrained links. While fundamental data structures such as source code representations are not necessarily expected to evolve gracefully over the course of a project, patterns of links constituting engineering processes and discourse should be.

We expect this to be a key concept in building highly adaptable knowledge navigation interfaces. The intermediate structural representations described above could be accumulated as a library of prototypical patterns of navigation. In a highly tangled web of resources, such patterns would constrain users' views, isolating patterns of discourse or engineering protocol that support their tasks. Bug reports, for example, may have regular patterns in their associated discussion and resolution. An interface could piece together these patterns to trace the genealogy (and associated rationale) behind a code unit. The discussion trees illustrated by Figure 6 provide a simple example of how navigational patterns would have simplified interface design. Rather than encode the client with specific knowledge of discussion structures, the system could have maintained a pattern describing general discourse relating to a resource.

Building a toolkit of common navigational patterns not only aids in rapid interface development, it facilitates intelligent prefetching for distributed objects. A server may use known patterns to anticipate resource traversals, automatically delivering commonly requested subgraphs.

8.2 Software Engineering Applications

The DATE framework provides a portable base for a number of interesting software engineering applications. The decomposition by definition applied to Common Lisp programs makes the archive

an excellent environment for managing *wrapper* code. We use this term to describe diagnostic code which operates before and after the invocation of a function or method, but would not be displayed in a normal source code editing view.

8.2.1 Monitoring Performance

A coherent method for managing performance metering and its results should be developed, especially if such code is to be rolled into a production system. Metering code during development helps engineers to localize bottlenecks and optimize algorithms. Less aggressive metering applied to a production system can be used to track emerging performance issues, whether due to security lapses or simply to changes in the operating environment. A system should be developed for determining the frequency and timing of such production-time monitoring for individual definitions, balancing the overhead of performing metering with the value of its results.

8.2.2 Statistical Profiling

In a system composed of fine-grained functional abstractions, determining which methods are typically invoked during production runs is as important as knowing their individual performance characteristics. With the DATE framework in place, an efficient system for capturing these statistics (at least for mid- to high-level abstractions) could be implemented in a straight-forward, portable manner.

8.2.3 Exception Handling

Engineering robust exception management can be one of the most challenging aspects of building a system, and error handling protocols tend to impose a great deal of complexity on source code. The abstractions expressed by method definitions, and therefore represented in the DATE framework, tend to be appropriate boundaries for containing exceptional conditions. We may use the archive to build a system for separating exception handling from the primary semantic threads of source code definitions. Interface support for this kind of functionality is an excellent example of the multidimensional editing concepts discussed in Section 4.1.2.

8.3 Archive Framework

8.3.1 Link Types

One area requiring a good deal of future work is the definition of link types. Just as developing archive structure and indexing adapts the system to a particular programming language, link type analysis accommodates different engineering processes and styles. Annotation links should be further categorized to capture discourse semantics, perhaps falling under different styles of conversation. Software engineering links can denote typical architectural relationships between regions of source code, e.g. producer/consumer relationships or dependencies on global state. Program monitoring services might utilize links describing algorithmic or performance invariants.

8.3.2 Resource Indexing

The Lisp code resource indexing implemented in Section 11.4.5 needs to be further abstracted. The current data structure built from hash tables and vectors should be replaced with a tree of CLOS classes representing sub-indexes. This will not only provide a cleaner interface, but significantly improve persistent object layer performance when saving archive objects. In the demonstration system, saving a top-level archive results in all resource name lookup tables being written to the instance log. Replacing Lisp primitives with persistent classes will allow incremental changes to a saved index, reducing the need for database compaction. Code unit names should also be replaced with instances of a new CLOS class, resulting in a more abstract, evolvable indexing interface.

8.3.3 Users

The DATE system needs to maintain a representation of the users of an archive. This would be used not only for authentication and permission purposes, but to automatically capture resource author information, and to initiate off-line communication such as email notifications. The system should utilize and extend the user model already defined as part of the CL-HTTP authentication system.

The system should also supply intelligent notification services, building upon an expert system tool such as Joshua [11]. Notification events might be triggered when a certain region of source code is modified, when a user is mentioned in an annotation, or when a bug is detected in a developer's code. Decisions governing if, when, and where to send such notifications should consider relevance to overall group goals, available notification mediums (email, pager, etc.), and user-configurable

priority rankings.

8.3.4 Low-Level Source Code Representations

While the core DATE archive is intended to provide only a platform for low-level source code encodings, developing such representations is an important extension. Regions of source code corresponding to versions (top-level Lisp forms) are currently represented as CRLF-encoded strings.

The first logical extension is to represent Lisp forms as nested list structures, as would be passed to `compile`. The major complication here is representing sharp-sign conditionalization within a definition. One plausible solution is to completely disallow such syntax, expanding existing conditionalized code into multiple implementations. A variant of this representation replaces globally bound symbols with actual code unit references. Immediate benefits include automatic, system-wide updates when changing the name of a function or variable (with the exception of certain run-time generated references).

8.3.5 Application to Programming Languages

Section 11.4.5 implements an archive for managing Common Lisp source code. Corresponding archive functionality should be added for languages such as Java or C++. As discussed in Section 3.3, Java source code would likely be decomposed into methods and instance variables, with classes acting as container resources. This representation may be useful in conjunction with the application of aspect-oriented programming techniques to Java. [15] [17].

8.4 Network Interface

8.4.1 Managing Clients

While the demonstration system includes resource locking capabilities, there exists little support for managing multiple clients simultaneously. The issue of primary importance is serializing the creation of new source code versions. Merging (or disallowing) parallel changes by multiple users should be handled at the network interface level, since the archive enforces linear version control. The network interface should also include resource change notification services, facilitating aggressive client-side caching.

8.4.2 Leveraging HTTP

The DATE network interface should make better use of HTTP/1.1 features offered by the CL-HTTP substrate. Archived DATE objects, in their serialized binary form, may be exported as individual HTTP resources. This immediately brings to bear the caching and authentication features included in the HTTP/1.1 specification. Proxy servers may be used as intermediate object caches, better servicing localized groups of developers. The server may invalidate or efficiently update resources by issuing HTTP cache directives. Using HTTP to cryptographically sign serialized objects helps ensure data integrity with minimal application-specific code.

8.5 User Interface

8.5.1 Constraint-Based Collections

The demonstration system supports only predefined collections of source code resources. It would be desirable to support more flexible, constraint-based views. An initial implementation could simply accept Lisp predicates that determine whether or not code units should be shown in the buffer. A more advanced version would provide a user interface and compute search constraints for efficiently accumulating resources. An example of a simple dynamic collection might be viewing all the definitions in a package that have active bug reports attached to them.

8.5.2 Code Unit Lookup

The system currently supports *meta-dot* for definition names that require only limited language semantics. This includes classes, functions, and methods but excludes nested definitions such as class accessors created by `defclass`. More interface support may be added as more language semantics are built upon the archive.

The interface is currently missing the facility to choose between multiple resources with the same name (but of different definition type). The ability to search by partial code unit name, or *apropos*, may also be added with no additional server-side support.

8.5.3 Archive Interface

The archive (and network) interface implemented in Section 11.7.2 insulates the rest of the client from knowledge of how objects are retrieved, but exposes the concept of object identifiers to other

Java classes. This design choice should be reevaluated, and knowledge of identifiers limited mostly to the `ArchiveInterface` class. This idea is partially implemented by the `getSlotObject` method. A similar wrapper method should be used for accessing objects residing in core Java data structures such as hash tables. This type of API is needed to develop better caching and prefetching abstractions.

8.6 Persistent Object Support

8.6.1 Tracking Class Evolution

As with many storage solutions, the persistent class system is fairly brittle with respect to class evolution. The system should, at minimum, provide the same level of support for class redefinition as CLOS. A better solution would allow developers to conveniently define mappings between class versions when rescuing old, incompatible data is considered important. These extensions imply storing versioned representations of classes in persistent storage alongside their instances.

8.6.2 Identifiers

Persistent identifiers should use global locators such as HTTP Universal Resource Identifiers (URI) to describe their instances. This will ease the transition to widely distributed development environments. The class and storage mechanism slots for identifiers currently use locally defined symbols.

8.6.3 Storage Mechanisms

Selecting the storage mechanism for saving a particular object should be a highly configurable, automated process. A generic function should be defined for optionally providing a default value other than `*default-storage-mechanism*`. A more powerful implementation might apply Joshua [11] to rank and select candidates.

Support for additional storage mechanisms should be added, using the log-based storage implementation in Section 11.3.7 as a model. A portable implementation using the ODBC protocol could be developed using existing support in several of the major Lisp implementations. For Macintosh Common Lisp (MCL), the Wood database would provide an appropriate local storage solution.

8.6.4 Root Objects

Root objects on storage mechanisms should be replaced by lookup services similar to that of a file system. A simple implementation would use hash tables exclusively as root objects, building a new object lookup interface around them.

8.7 Distributed Object Management

8.7.1 Java RMI

While we have successfully built a Java-to-Lisp Remote Method Invocation (RMI) interface, it is desirable to have Lisp-to-Java RMI support, as well. Since our Common Lisp implementation of Java object serialization is already functional, the main task is implementing a Lisp client for the Java RMI server [9]. This will allow the server to initiate communication with remote clients in addition to servicing their requests. While mapping CLOS classes to Java results in some incompatibilities, the inheritance and instance variable structure of Java classes may be fully represented in Lisp. Since serialization streams contain such class data, Common Lisp would immediately have the ability to communicate with most existing Java RMI servers. Understanding results requires semantic knowledge of Java class structures, but intermediate RMI results could automatically be represented and resubmitted as arguments. For example, a Lisp client (without any knowledge of the data structures within a class) might retrieve an image from one server, submit it for processing at several others, and send the results to a final server.

8.7.2 CLOS Inheritance

The translation of CLOS inheritance structures to Java could also be improved. The current implementation flattens inheritance trees, accumulating all accessible slots into a single Java class. This is sufficient for transporting data, but important abstractions are lost. Alternative solutions, perhaps utilizing Java interfaces, should be considered.

9 Related Research

Research focusing on software engineering and tools to support collaborative system building includes a broad spectrum of goals and approaches. The following subsections describe research that either has similarity to the system outlined in this paper, or helped to inspire the need for such a system.

9.1 Gwydion Project

In a 1994 white paper entitled *Gwydion: An Integrated Software Environment for Evolutionary Software Development and Maintenance* [3], Scott E. Fahlman outlines the motivations and goals of a new software engineering initiative at Carnegie Mellon University. The author states the objectives for an environment that would cleanly differentiate the development and production phases of software development such that dynamic debugging can be achieved without sacrificing efficient executable code. He asserts that the Dylan language is appropriate for this role, providing both a dynamic development environment and compiled production executables that compete with C and Fortran in efficiency.

While the Gwydion project was intended to create a complete, high-quality development system as a deliverable, it shared several top-level goals with the project at hand. Fahlman stresses the importance of moving from a linear source code representation to a more complex, linked data structure. While this approach is not unique in software development environments, both the Gwydion project and DATE propose that this structure should be a platform for programmer support services throughout the project life-cycle. In Fahlman's description of hypercode, he discusses the importance of multiple, customizable source code views. This approach could apply to multiple levels of abstraction, or to different (possibly overlapping) logical groupings of code. The paper also refers to software librarians and consistency monitors that will utilize the program database structure. DATE shares the Gwydion view of rich source code representations, but attempts to take a broader view. in order to address more general issues such as documentation maintenance and project management, we build abstractions that do not depend on detailed knowledge of language semantics .

In 1996, after Apple Computer ceased development work on the Dylan language, the Gwydion project shifted their efforts to a similar Java development system. The project was later terminated due to funding constraints. The existing Gwydion software was moved into the public domain, and there now exists a downloadable version of Sheets, the Java version of the project's development environment.

9.2 Global Cooperative Computing

At the Second International WWW Conference in 1994, André DeHon *et al.* [2] presented their vision of a wide-area collaboration system for software development. The authors enumerate a number of the qualities that characterize a modern source code representation, including annotation and quality assertions, links to related sources of information, and "program genealogy" structures. The scenarios portrayed are especially applicable to open-source, distributed collaboration efforts such as developing free operating systems. The authors also discuss applying such a system to the automatic collection and correlation of implicit program feedback, such as the frequency of fatal errors, usage of features, and session length. The type of collaboration system described is very similar to that implemented in the Open Meeting system [7], developed by the Intelligent Information Infrastructure Project at the MIT AI Laboratory. DATE aims to support and extend this type of engineering discourse, but built upon a framework better suited to supporting the evolution of running systems in dynamic programming environments.

9.3 Software Architecture

Mary Shaw, David Garlan, and others at Carnegie Mellon University have worked to refine and promote a systematic approach to software architecture. Much of their published work has focused on modularity, interconnection, and component reuse as emerging areas of computer science akin to algorithms or data structures. They identify standard architectural styles [5], formalize a description language for module interconnection [19], and develop tools for describing systems at the architectural level [6].

If one considers the area of software architecture with system evolvability as a key design goal, the need for a flexible source code representation becomes more apparent. We wish to support an architectural view of a system not only during its initial design, but throughout the software lifecycle in order to adapt to changing requirements and resource constraints. The power of a visual architectural design tool such as the Aesop system [6] could be brought fully to bear with concrete links to the DATE resources it describes. Some of these issues were considered at a somewhat lower level by Richard Waters and Yang Meng Tan [23] at the MIT Artificial Intelligence Laboratory. The common problem of documentation consistency will also apply to architectural descriptions of software. The evolution of a large system will likely include architectural changes, coercing existing functionality into a fundamentally different architectural style. Consider, for example, the large

number of conventional applications that have been ported to the client-server model of the Web.

10 References

[1] Brooks, Frederick P. *The Mythical Man-Month*, anniversary ed., Addison-Wesley, 1995.

[2] DeHon, André, Jeremy Brown, Ian Eslick, Jake Harris, Lara Karbiner, Thomas F. Knight, Jr. "Global Cooperative Computing," *Proceedings of the Second International WWW Conference*, Chicago, Oct 1994.

[3] Fahlman, Scott. E. *Gwydion: An Integrated Software Environment for Evolutionary Software Development and Maintenance.* Carnegie Mellon University: Pittsburg, PA. Mar 26, 1994. `http://www.cs.cmu.edu/afs/cs/project/gwydion/docs/htdocs/gwydion-overview.html`

[4] Fielding, R., J. Gettys *et al. Hypertext Transfer Protocol – HTTP/1.1*, HTTP Working Group, W3C, 18 Nov 1998. `http://www.w3.org/Protocols/HTTP/1.1/draft-ietf-http-v11-spec-rev-06.txt`

[5] Garlan, David & Mary Shaw. "An Introduction to Software Architecture," *Advances in Software Engineering and Knowledge Engineering, Volume I*, eds. V. Ambriola & G. Tortora, World Scientific Publishing Company, New Jersey, 1993.

[6] Garlan, David, Robert Allen & John Ockerbloom. "Exploiting Style in Architectural Design Environments," *Proceedings of the ACM SIGSOFT '94 Symposium on Foundations of Software Engineering*, New Orleans, Dec 1994.

[7] Hurwitz, Roger & John C. Mallery, "The Open Meeting: A Web-Based System for Conferencing and Collaboration," *World Wide Web Journal*, 1(1), Winter, 1996.

[8] *Java 2 Platform API Specification*, Sun Microsystems, Inc., 1999. `http://java.sun.com/products/jdk/1.2/docs/api/index.html`

[9] *Java Remote Method Invocation Specification*, Sun Microsystems, Inc., Oct 1998. `http://java.sun.com/products/jdk/1.2/docs/guide/rmi/`

[10] *Java Object Serialization Specification*, Sun Microsystems, Inc., 1998. `http://java.sun.com/products/jdk/1.2/docs/guide/serialization/`

[11] *Joshua Reference Manual*, Symbolics, Inc., Feb 1990.

[12] Katz, Boris. "From Sentence Processing to Information Access on the World Wide Web," *Natural Language Processing for the World Wide Web*, AAAI Spring Symposium, Apr 1997.

[13] Keene, Sonya E. *Object-Oriented Programming in Common Lisp*, Addison-Wesley, 1989.

[14] Kiczales, Gregor, Jim Des Rivieres, and Daniel Bobrow. *The Art of the Metaobject Protocol*, MIT Press, 1991.

[15] Kiczales, Gregor, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Videira Lopes, Jean-Marc Loingtier, and John Irwin. "Aspect-Oriented Programming," *Proceedings of the European Conference on Object-Oriented Programming (ECOOP)*, Finland, Springer-Verlag LNCS 1241, Jun 1997.

[16] Lassila, Ora & Ralph R. Swick, eds. *Resource Description Framework (RDF) Model and Syntax*, W3C Working Draft, Feb 16, 1998.

[17] Lopes, Cristina Videira and Gregor Kiczales, "Recent Developments in AspectJ?," *ECOOP'98 Workshop Reader*, Springer-Verlag LNCS 1543, 1998.

[18] Mallery, John C., "A Common LISP Hypermedia Server," *Proceedings of the First International WWW Conference*, Geneva, May 25, 1994.

[19] Shaw, Mary, Robert DeLine, Daniel V. Klein, Theodore L. Ross, David M. Young & Gregory Zelesnik. "Abstractions for Software Architecture and Tools to Support Them," *IEEE Transactions on Software Engineering*, vol 21, no 4, Apr 1995.

[20] Solderitsch, James, Elizabeth A. Bradley, & Timothy M. Schreyer. "The Reusability Library Framework - Leveraging Software Reuse," *Paramax System and Software Engineering Symposium*, Nov 1992.
`http://www.asset.com/stars/lm-tds/Papers/sses-rlf/symp.html`

[21] Stallman, Richard M. *GNU Emacs Manual*, Ninth Ed., Version 19, Free Software Foundation, Inc., Aug 1993.

[22] Steele, Guy L. Jr. *Common Lisp the Language, Second Edition*, Butterworth-Heinemann, 1990.

[23] Tan Yang Meng. *Supporting Reuse and Evolution in Software Design*, MIT AI Laboratory, Oct 1990.

[24] Uzzle, Lyn. "SEEWeb: A Software Information Interface Based on World Wide Web Technology," *Software Technology Conference (STC '96)*, Salt Lake City, Apr 1996.
`http://www.asset.com/stars/lm-tds/Papers/seeweb/seewebtoc.html`

11 Source Code

11.1 Shared Utilities

11.1.1 code/cds-util/package.lisp

```
;;; -*- Mode: lisp; Syntax: ANSI-Common-Lisp; Base: 10; Package: cl-user; -*-
;;;
;;; Copyright 1998-99 Christopher R. Vincent (cvince@mit.edu)
;;; All Rights Reserved.
;;;
;;;---------------------------------------------------------------------
;;;
;;; CDS UTILITIES
;;;

(in-package :cl-user)

(defpackage collaborative-development-server-utils
  (:nicknames cds-util)
  (:use future-common-lisp task-queue)
  (:import-from "HTTP"
   "INTERN-KEYWORD")
  (:import-from "RESOURCES"
   "DEFRESOURCE"
   "USING-RESOURCE")
  (:import-from "WWW-UTILS"
   "ATOMIC-DECF"
   "ATOMIC-INCF"
   "ATOMIC-POP"
   "ATOMIC-PUSH"
   "ATOMIC-SETF"
   "DIRECTORY-LIST*"
   "MAKE-LOCK"
   "WITH-ATOMIC-EXECUTION"
   "WITH-LOCK-HELD")
  (:export
   "ATOMIC-DECF"
   "ATOMIC-INCF"
   "ATOMIC-POP"
   "ATOMIC-PUSH"
   "ATOMIC-SETF"
   "CLEAR-TASK-QUEUE"
   "DEFRESOURCE"
   "DIRECTORY-LIST*"
   "*EMPTY-PACKAGE*"
   "ENSURE-ACTIVE-TASK-QUEUE"
   "INTERN-KEYWORD"
   "MAKE-LOCK"
   "MAKE-WEAK-HASH-TABLE"
   "PUSH-TASK-QUEUE"
   "SET-WEAK-HASH-TABLE"
   "SHORT-PACKAGE-NAME"
   "STOP-TASK-QUEUE"
   "TASK-QUEUE"
   "USING-RESOURCE"
   "WEAK-GETHASH"
   "WEAK-MAPHASH"
   "WEAK-CLRHASH"
   "WHEN-BIND"
   "WITH-ATOMIC-EXECUTION"
   "WITH-LOCK-HELD"
   "WRITE-TIME-TO-STRING"))
```

11.1.2 code/cds-util/util.lisp

```
;;; -*- Mode: lisp; Syntax: ANSI-Common-Lisp; Base: 10; Package: cds-util; -*-
;;;
;;; Copyright 1998-99 Christopher R. Vincent (cvince@mit.edu)
;;; All Rights Reserved.
;;;
;;;---------------------------------------------------------------------
;;;
;;; PACKAGE UTILITIES
;;;

(in-package :cds-util)

(defvar *package-nicknames* (make-hash-table)
  "Keep track of the shortest name for a package.")

(defun short-package-name (package)
  (cond ((gethash package *package-nicknames*))
        (t (let ((nicknames (package-nicknames package)))
             (cond ((null nicknames)
                    (setf (gethash package *package-nicknames*) (package-name package)))
                   (t (loop for name in (rest nicknames)
                            with short = (car nicknames)
                            with short-len = (length short)
                            for name-len = (length name)
                            when (< name-len short-len)
                            do (setq short name
                                     short-len name-len)
                            finally (return (setf (gethash package *package-nicknames*) sh\
ort)))))))))

;; Need a package that is guaranteed empty for printing symbols.
(defpackage empty)

(defparameter *empty-package* (find-package :empty)
  "The empty package, may be used for printing package-qualified symbols.")

(defmacro when-bind ((var form) &body body)
  "Bind a value and execute body when it is not NIL."
  `(let ((,var ,form))
     (when ,var
       ,@body)))

(defun write-time-to-string (&optional (universal-time (get-universal-time)))
  (with-output-to-string (stream)
    (http::write-standard-time universal-time stream)))
```

11.1.3 code/platform/mcl/cds-util/package.lisp

```
;;; -*- Mode: lisp; Syntax: ANSI-Common-Lisp; Base: 10; Package: cds-util; -*-
;;;
;;; Copyright 1998-99 Christopher R. Vincent (cvince@mit.edu)
;;; All Rights Reserved.
;;;
;;;---------------------------------------------------------------------
;;;
;;; UTIL
;;;

(in-package :cds-util)

(import '(ccl:class
          ccl:class-direct-superclasses
          ccl:class-precedence-list
          ccl::lock
```

```
          ccl:slot-definition-name
          ccl:slot-definition-type)
        :cds-util)

(defmacro class-slots (ccl:class)
  `(concatenate 'list (ccl:class-instance-slots ,ccl:class) (ccl:class-class-slots ,ccl:cl\
ass)))

(export '(ccl:class
          ccl:class-direct-superclasses
          ccl:class-precedence-list
          class-slots
          ccl::lock
          ccl:slot-definition-name
          ccl:slot-definition-type)
        :cds-util)
```

11.1.4 code/platform/mcl/cds-util/weak-hash-table.lisp

```
;;; -*- Syntax: ANSI-Common-Lisp; Base: 10; Package: CDS-UTIL -*-
;;;
;;; Copyright 1998-99 Christopher R. Vincent (cvince@mit.edu)
;;; All Rights Reserved.
;;;
;;;---------------------------------------------------------------------
;;;
;;; WEAK HASH TABLES
;;;

(in-package :cds-util)

;; This still needs locking(?)

(defun make-weak-hash-table (&key (test #'eql))
  (make-hash-table :test test :weak t))

(setf (symbol-function 'weak-gethash) (symbol-function 'gethash)
      (symbol-function 'weak-maphash) (symbol-function 'maphash)
      (symbol-function 'weak-clrhash) (symbol-function 'clrhash))

(defun set-weak-hash-table (key weak-hash-table value)
  (setf (gethash key weak-hash-table) value))

(defsetf weak-gethash set-weak-hash-table)
```

11.2 Java Object Serialization Protocol

11.2.1 code/java/package.lisp

```
;;; -*- Mode: lisp; Syntax: ANSI-Common-Lisp; Base: 10; Package: cl-user; -*-
;;;
;;; Copyright 1998-99 Christopher R. Vincent (cvince@mit.edu)
;;; All Rights Reserved.
;;;
;;;---------------------------------------------------------------------
;;;
;;; JAVA OBJECT SERIALIZATION PROTOCOL
;;;

(in-package :cl-user)
```

```
(defpackage java-object-serialization
  (:nicknames jos)
  (:use future-common-lisp cds-util)
  (:export
   "FLUSH-CACHED-CLASS-MAPPINGS"
   "GENERATE-JAVA-SOURCES"
   "INPUT-BYTE-COUNT"
   "INSTANCE-FOR-SERIALIZATION"
   "OUTPUT-BYTE-COUNT"
   "READ-OBJECT"
   "WITH-JAVA-SERIALIZATION-INPUT"
   "WITH-JAVA-SERIALIZATION-OUTPUT"
   "WRITE-OBJECT"))
```

11.2.2 code/java/serialization.lisp

```
;;; -*- Mode: lisp; Syntax: ANSI-Common-Lisp; Base: 10; Package: JOS -*-
;;;
;;; Copyright 1998-99 Christopher R. Vincent (cvince@mit.edu)
;;; All Rights Reserved.
;;;
;;;------------------------------------------------------------------
;;;
;;; JAVA OBJECT SERIALIZATION PROTOCOL
;;;

#|

A simple, fast, mapping between Lisp and Java objects using the
java.io.Serializable interface.

While the mapping is not lossless, it should be sufficient for many
Java interfaces.  More Lisp types could easily be added to the lisp.lang
package (see lisp.lang.Symbol).  Inheritance for CLOS classes is only used
a programming convenience on the Java end, parents are defined arbitrarily
when sources are generated.  We could use a more precise mapping, handle
inheritance right.  Then again, you could just use CORBA.

Lisp    <-->   Java

t                java.lang.Boolean
nil              null
integer          java.lang.Byte
                 java.lang.Short
                 java.lang.Integer
                 java.lang.Long
float/rational   java.lang.Float
                 java.lang.Double
character        java.lang.Character
string           java.lang.String
vector           array
cons             java.util.Vector
hash-table       java.util.Hashtable
symbol           lisp.lang.Symbol
keyword          lisp.lang.Keyword
instance         lisp.map.*


Issues:
Handle multi-dimensional arrays.
Ensure that field names are always sorted in the same order as in Java.
Fix write-float, implement double.
Current name doesn't worry about Java name length, a problem with some filesystems.
Could use classDesc info to implement less brittle class mappings (check slot names).
Better blockdata handling, environments should have a byte counter.
```

```
read-classdata should allow the instance creator to be overridden.
write-class/object-annotation isn't used, need to differentiate between
  blockdata content and normal Lisp vectors.  Probably CLOSify blockdata.

|#

(in-package :jos)

;;;-------------------------------------------------------------------
;;;
;;; CONSTANTS
;;;

(eval-when (compile load eval)

  ; Typecodes from specification.
  (defconstant +prim-typecode-byte+ (char-code #\B))
  (defconstant +prim-typecode-char+ (char-code #\C))
  (defconstant +prim-typecode-double+ (char-code #\D))
  (defconstant +prim-typecode-float+ (char-code #\F))
  (defconstant +prim-typecode-integer+ (char-code #\I))
  (defconstant +prim-typecode-long+ (char-code #\J))
  (defconstant +prim-typecode-short+ (char-code #\S))
  (defconstant +prim-typecode-boolean+ (char-code #\Z))
  (defconstant +obj-typecode-array+ (char-code #\[))
  (defconstant +obj-typecode-object+ (char-code #\L))

  ; Terminal and constant values from specification.
  (defconstant +stream-magic+ #xACED)
  (defconstant +stream-version+ 5)
  (defconstant +tc-null+ #x70)
  (defconstant +tc-reference+ #x71)
  (defconstant +tc-classdesc+ #x72)
  (defconstant +tc-object+ #x73)
  (defconstant +tc-string+ #x74)
  (defconstant +tc-array+ #x75)
  (defconstant +tc-class+ #x76)
  (defconstant +tc-blockdata+ #x77)
  (defconstant +tc-endblockdata+ #x78)
  (defconstant +tc-reset+ #x79)
  (defconstant +tc-blockdatalong+ #x7A)
  (defconstant +tc-exception+ #x7B)
  (defconstant +base-wire-handle+ #x7E0000)
  (defconstant +sc-write-method+ #x01)          ; if +sc-serializable+
  (defconstant +sc-block-data+ #x08)            ; if +sc-externalizable+
  (defconstant +sc-serializable+ #x02)
  (defconstant +sc-externalizable+ #x04)

  ; Lisp integer manipulation.
  (defconstant +byte-4-0+ (byte 4 0))
  (defconstant +byte-5-0+ (byte 5 0))
  (defconstant +byte-6-0+ (byte 6 0))
  (defconstant +byte-8-0+ (byte 8 0))
  (defconstant +byte-23-0+ (byte 23 0))
  (defconstant +byte-2-6+ (byte 2 6))
  (defconstant +byte-5-6+ (byte 5 6))
  (defconstant +byte-7-6+ (byte 7 6))
  (defconstant +byte-1-7+ (byte 1 7))
  (defconstant +byte-8-8+ (byte 8 8))
  (defconstant +byte-4-12+ (byte 4 12))
  (defconstant +byte-8-16+ (byte 8 16))
  (defconstant +byte-8-23+ (byte 8 23))
  (defconstant +byte-8-24+ (byte 8 24))
  (defconstant +byte-1-31+ (byte 1 31))
  (defconstant +byte-8-32+ (byte 8 32))
  (defconstant +byte-8-40+ (byte 8 40))
  (defconstant +byte-8-48+ (byte 8 48))
  (defconstant +byte-8-56+ (byte 8 56))
```

```
  )

;;;---------------------------------------------------------------------
;;;
;;; DEBUGGING
;;;

(defconstant +debug-serialization+ nil)

(defvar *debugging-depth* 0)

(defmacro debug-serialization (string &rest args)
  (cond (+debug-serialization+
         `(progn
            (fresh-line t)
            (loop repeat (* 3 *debugging-depth*) do (write-char #\Space t))
            (format t ,string ,@args)))
        (t string args  ; ignore
           nil)))

(defmacro with-serialization-debugging ((string &rest args) &body body)
  (cond (+debug-serialization+
         `(let ((*debugging-depth* (or *debugging-depth* 0)))
            (declare (special *debugging-depth*))
            (fresh-line t)
            (loop repeat (* 3 *debugging-depth*) do (write-char #\Space t))
            (format t ,string ,@args)
            (incf *debugging-depth*)
            (let ((result (progn ,@body)))
              (decf *debugging-depth*)
              result)))
        (t string args ; ignore
           `(progn ,@body))))

;;;---------------------------------------------------------------------
;;;
;;; UTILITIES
;;;

(defmacro with-input-from-vector ((stream-var vector) &body body)
  `(let ((,stream-var (cons ,vector 0)))
     ,@body))

(defmacro with-output-to-vector ((stream-var) &body body)
  `(let ((,stream-var (make-array 0 :fill-pointer t :adjustable t)))
     ,@body
     ,stream-var))

(defvar *bytes-written* 0)

(defvar *bytes-read* 0)

(defmethod write-unsigned-byte (number (output vector))
  (incf *bytes-written*)
  (vector-push-extend number output))

(defmethod write-unsigned-byte (number (output stream))
  (incf *bytes-written*)
  (write-byte number output))

(defun write-unsigned-bytes (vector stream)
  (loop for b across vector
        do (write-unsigned-byte b stream)))

(defmethod read-unsigned-byte ((input cons))
  (incf *bytes-read*)
  (prog1
    (elt (car input) (cdr input))
    (incf (cdr input))))
```

```
(defmethod read-unsigned-byte ((input stream))
  (incf *bytes-read*)
  (read-byte input nil nil))

(defun write-signed-byte (number stream)
 (write-unsigned-byte (logand #xff number) stream))

(defun write-signed-bytes (vector stream)
  (loop for b across vector
        do (write-signed-byte b stream)))

(defun read-signed-byte (stream)
  (let ((n (read-unsigned-byte stream)))
    (cond ((logbitp 7 n) (- (logand #xff (lognot (1- n)))))
          (t n))))

(defun write-unsigned-short (number stream)
  (write-unsigned-byte (ldb +byte-8-8+ number) stream)
  (write-unsigned-byte (ldb +byte-8-0+ number) stream))

(defun read-unsigned-short (stream)
  (logior (ash (read-unsigned-byte stream) 8)
          (read-unsigned-byte stream)))

(defun write-signed-short (number stream)
  (write-unsigned-short (logand #xffff number) stream))

(defun read-signed-short (stream)
  (let ((n (read-unsigned-short stream)))
    (cond ((logbitp 15 n) (- (logand #xffff (lognot (1- n)))))
          (t n))))

(defun write-unsigned-int (number stream)
  (write-unsigned-byte (ldb +byte-8-24+ number) stream)
  (write-unsigned-byte (ldb +byte-8-16+ number) stream)
  (write-unsigned-byte (ldb +byte-8-8+ number) stream)
  (write-unsigned-byte (ldb +byte-8-0+ number) stream))

(defun write-signed-int (number stream)
  (write-unsigned-int (logand #xffffffff number) stream))

(defun read-unsigned-int (stream)
  (logior (ash (read-unsigned-byte stream) 24)
          (ash (read-unsigned-byte stream) 16)
          (ash (read-unsigned-byte stream) 8)
          (read-unsigned-byte stream)))

(defun read-signed-int (stream)
  (let ((n (read-unsigned-int stream)))
    (cond ((logbitp 31 n) (- (logand #xffffffff (lognot (1- n)))))
          (t n))))

(defun write-unsigned-long (number stream)
  (write-unsigned-byte (ldb +byte-8-56+ number) stream)
  (write-unsigned-byte (ldb +byte-8-48+ number) stream)
  (write-unsigned-byte (ldb +byte-8-40+ number) stream)
  (write-unsigned-byte (ldb +byte-8-32+ number) stream)
  (write-unsigned-byte (ldb +byte-8-24+ number) stream)
  (write-unsigned-byte (ldb +byte-8-16+ number) stream)
  (write-unsigned-byte (ldb +byte-8-8+ number) stream)
  (write-unsigned-byte (ldb +byte-8-0+ number) stream))

(defun write-signed-long (number stream)
  (write-unsigned-long (logand #xffffffffffffffff number) stream))

(defun read-unsigned-long (stream)
  (logior (ash (read-unsigned-byte stream) 56)
          (ash (read-unsigned-byte stream) 48)
```

```
          (ash (read-unsigned-byte stream) 40)
          (ash (read-unsigned-byte stream) 32)
          (ash (read-unsigned-byte stream) 24)
          (ash (read-unsigned-byte stream) 16)
          (ash (read-unsigned-byte stream) 8)
          (read-unsigned-byte stream)))

(defun read-signed-long (stream)
  (let ((n (read-unsigned-long stream)))
    (cond ((logbitp 63 n) (- (logand #xffffffffffffffff (lognot (1- n)))))
          (t n))))

; This should implement IEEE 754 floating-point correctly!
; Close enough for now.
(defun write-float (number stream)
  (let* ((float (float number))
         (digits (float-digits float))
         (shift-count (max 0 (- digits 24))))
    (multiple-value-bind (significand exponent sign) (integer-decode-float float)
      (let ((biased-exponent (ldb +byte-8-0+ (+ exponent shift-count 150))))
        (setq significand (if (zerop biased-exponent)
                              (ldb (byte 23 (1+ shift-count)) significand)
                              (ldb (byte 23 shift-count) significand)))
        (write-unsigned-int (logior (ash (if (> sign 0) 0 1) 31)
                                    (ash biased-exponent 23)
                                    significand)
                            stream)))))

(defun read-float (stream)
  (let ((bits (read-unsigned-int stream)))
    (cond ((= #x7f800000 bits) most-positive-long-float)
          ((= #xff800000 bits) most-negative-long-float)
          ((and (>= bits #x7f800001) (<= bits #x7fffffff)) nil)  ; NaN
          ((and (>= bits #xff800001) (<= bits #xffffffff)) nil)
          (t (let* ((s (if (zerop (ldb +byte-1-31+ bits)) 1 -1))
                    (e (ldb +byte-8-23+ bits))
                    (m (if (zerop e)
                           (ash (ldb +byte-23-0+ bits) 1)
                           (logior #x800000 (ldb +byte-23-0+ bits)))))
               (* s m (expt (float 2) (- e 150))))))))

(defun write-utf-string (string stream &optional utf-length)
  (flet ((scan-length ()
           (loop for c across string
                 for code = (char-code c)
                 sum (cond ((zerop code) 2)
                           ((< code #x80) 1)
                           ((< code #x7FF) 2)
                           (t 3))))
         (write-utf-char (code)
           (cond ((zerop code)
                  ; Java variant of UTF-8 encodes null as two bytes.
                  (write-unsigned-byte #xE0 stream)
                  (write-unsigned-byte #x80 stream))
                 ((< code #x80)
                  (write-unsigned-byte code stream))
                 ((< code #x7FF)
                  (write-unsigned-byte (logior #xE0 (ldb +byte-5-6+ code)) stream)
                  (write-unsigned-byte (logior #x80 (ldb +byte-6-0+ code)) stream))
                 (t (write-unsigned-byte (logior #xE0 (ldb +byte-4-12+ code)) stream)
                    (write-unsigned-byte (logior #xE0 (ldb +byte-7-6+ code)) stream)
                    (write-unsigned-byte (logior #x80 (ldb +byte-6-0+ code)) stream)))))
    (declare (inline scan-length write-utf-char))
    (write-unsigned-short (or utf-length (scan-length)) stream)
    (loop for idx from 0 upto (1- (length string))
          do (write-utf-char (char-code (char string idx))))))

(defun read-utf-string (stream)
  (loop with chars = (make-array 0 :fill-pointer t :adjustable t)
```

```
        and length = (read-unsigned-short stream)
        and saved
        for byte0 = (cond (saved (prog1 saved (setq saved nil)))
                          (t (read-unsigned-byte stream)))
        when (zerop (ldb +byte-1-7+ byte0))
        do (vector-push-extend (character byte0) chars)
        (decf length)
        else do
        (let ((byte1 (read-unsigned-byte stream))
              byte2)
          (cond ((and (>= length 3)
                      (eq 2 (ldb +byte-2-6+ (setq byte2 (read-unsigned-byte stream)))))
                 (vector-push-extend (character
                                      (logior (ash (ldb +byte-4-0+ byte0) 12)
                                              (ash (ldb +byte-5-0+ byte1) 6)
                                              (ldb +byte-6-0+ byte2)))
                                     chars)
                 (decf length 3))
                (t (setq saved byte2)
                   (vector-push-extend (character
                                        (logior (ash (ldb +byte-5-0+ byte0) 6)
                                                (ldb +byte-6-0+ byte1)))
                                       chars)
                   (decf length 2))))
        while (> length 0)
        finally (return (coerce chars 'string))))

(defun cons-length (cons)
  "Compute the length of an improper list, (1+ (length cons)) for normal lists."
  (loop for count upfrom 0
        for p = cons then (cdr p)
        until (not (consp p))
        finally (return (1+ count))))

;;;---------------------------------------------------------------------
;;;
;;; JAVA NAME MAPPING
;;;

(defun symbol-to-java-name (symbol &key capitalize-p package-p)
  (flet ((convert-string (string &optional upcase-p)
           (loop for c across string
                 with upcase = upcase-p
                 when (eq #\- c)
                 do (setq upcase t)
                 else when upcase
                 collect (char-upcase c) into result
                 and do (setq upcase nil)
                 else collect c into result
                 finally (return (coerce result 'string)))))
    (declare (inline convert-string))
    (let ((name (convert-string (string-downcase (symbol-name symbol)) capitalize-p))
          package
          (result nil))
      (when (and package-p (setq package (symbol-package symbol)))
        (let ((short-name (string-downcase (short-package-name package))))
          (setq result (concatenate 'string (convert-string short-name t) "_"))))
      (concatenate 'string result name))))

(defun java-name-to-symbol (java-name &key (default-package *package*))
  (loop for c across java-name
        with downcase-min = (char-code #\a)
        with package
        with name
        with idx = 0
        when (eq #\_ c)
        do (setq package (coerce (nreverse name) 'string)
                 name nil
                 idx 0)
```

```
        else when (and (< (char-code c) downcase-min)
                       (not (zerop idx)))
        do (push #\- name)
        (push c name)
        (incf idx)
        else
        do (push (char-upcase c) name)
        (incf idx)
        finally (return (intern (coerce (nreverse name) 'string)
                                (or package default-package)))))

(defun symbol-to-java-class-name (symbol)
  (let ((class-name (symbol-to-java-name symbol :capitalize-p t :package-p t)))
    (concatenate 'string "lisp.map." class-name)))

(defun java-class-name-to-symbol (java-name)
  (java-name-to-symbol java-name))

;;;---------------------------------------------------------------------
;;;
;;; CLASS DESCRIPTIONS
;;;

(defclass class-desc ()
  ((name
    :initarg :name
    :accessor class-desc-name)
   (serial-version-uid
    :initarg :serial-version-uid
    :accessor class-desc-serial-version-uid)
   (flags
    :initarg :flags
    :accessor class-desc-flags)
   (fields
    :initarg :fields
    :accessor class-desc-fields
    :documentation "List of (typecode fieldName className1) triplets.")
   (class-annotation
    :initarg :class-annotation
    :accessor class-desc-class-annotation)
   (super-class
    :initarg :super-class
    :accessor class-desc-super-class))
  (:documentation "Information pertaining to class descriptions."))

(defclass java-class-desc (class-desc)
  ((object-writer
    :initarg :object-writer
    :accessor class-desc-object-writer
    :documentation "Function for writing complete newObject entries.")
   (classdata-reader
    :initarg :classdata-reader
    :accessor class-desc-classdata-reader
    :documentation "Function for reading the classdata portion of newObject entries.
                    Responsible for allocating newHandle is an object is created."))
  (:documentation "A known Java class."))

(defclass lisp-class-desc (class-desc)
  ((class-name
    :initarg :class-name
    :accessor class-desc-class-name)
   (slot-names
    :initarg :slot-names
    :accessor class-desc-slot-names))
  (:documentation "A Java mapping for a Lisp class."))

(defvar *class-desc-table* (make-hash-table :test #'equal)
  "Maps Lisp classes and Java class names to class-desc instances.")
```

```
(defun define-java-class (name &key super-class
                               serial-version-uid
                               (class-desc-flags +sc-serializable+)
                               fields
                               class-annotation
                               object-writer
                               classdata-reader)
  "Register information about an existing Java class."
  (setf (gethash name *class-desc-table*)
        (make-instance 'java-class-desc
          :name name
          :serial-version-uid serial-version-uid
          :flags class-desc-flags
          :fields fields
          :class-annotation class-annotation
          :super-class (when super-class (gethash super-class *class-desc-table*))
          :object-writer object-writer
          :classdata-reader classdata-reader)))

(defun define-lisp-mapping (class)
  (let* ((class-name (class-name class))
         (name (symbol-to-java-class-name class-name))
         (slot-names (mapcar #'slot-definition-name (class-slots class)))
         (package (symbol-package class-name))
         (field-specs (stable-sort
                       (mapcar #'(lambda (slot-name)
                                   (cons (symbol-to-java-name
                                          slot-name :package-p (not (eq package (symbol-pa\
ckage slot-name))))
                                         slot-name))
                               slot-names)
                       #'(lambda (a b) (string< (car a) (car b)))))
         sorted-slot-names
         (fields (loop for (field . slot) in field-specs
                       do (setq sorted-slot-names (nconc sorted-slot-names (list slot)))
                       collect (list +obj-typecode-object+ field "Ljava/lang/Object;")))
         (class-desc (make-instance 'lisp-class-desc
                       :name name
                       :serial-version-uid 1  ; Always use the same serial-version-uid.
                       :flags +sc-serializable+
                       :fields fields
                       :class-annotation nil
                       :super-class nil
                       :class-name class-name
                       :slot-names sorted-slot-names)))
    (setf (gethash class *class-desc-table*) class-desc
          (gethash name *class-desc-table*) class-desc)))

(defun get-class-desc (class)
  (or (gethash class *class-desc-table*)
      (etypecase class
        (class (let ((class-desc (define-lisp-mapping class)))
                 (setf (gethash class *class-desc-table*) class-desc
                       (gethash (class-desc-name class-desc) *class-desc-table*) class-des\
c)))
        ; Java class string, attempt to find corresponding Lisp class.
        (string (let* ((name-start (1+ (position #\. class :from-end t)))
                       (class-name (java-class-name-to-symbol (subseq class name-start)))
                       (lisp-class (find-class class-name))
                       (class-desc (define-lisp-mapping lisp-class)))
                  (setf (gethash lisp-class *class-desc-table*) class-desc
                        (gethash class *class-desc-table*) class-desc))))))

;;;-------------------------------------------------------------------
;;;
;;; KNOWN JAVA CLASSES
;;;

(defun write-java-boolean (p stream)
```

```
  (write-unsigned-byte +tc-object+ stream)
  (write-class-desc (get-class-desc "java.lang.Boolean") stream)
  (allocate-new-handle p)
  (write-signed-byte (if p 1 0) stream))

(defun read-java-boolean-classdata (stream)
  (= 1 (read-signed-byte stream)))

(defun write-java-character (c stream)
  (write-unsigned-byte +tc-object+ stream)
  (write-class-desc (get-class-desc "java.lang.Character") stream)
  (allocate-new-handle c)
  (write-unsigned-short (char-code c) stream))

(defun read-java-character-classdata (stream)
  (character (read-signed-short stream)))

(defun write-java-byte (number stream)
  (write-unsigned-byte +tc-object+ stream)
  (write-class-desc (get-class-desc "java.lang.Byte") stream)
  (allocate-new-handle number)
  (write-signed-byte number stream))

(defun write-java-short (number stream)
  (write-unsigned-byte +tc-object+ stream)
  (write-class-desc (get-class-desc "java.lang.Short") stream)
  (allocate-new-handle number)
  (write-signed-short number stream))

(defun write-java-integer (number stream)
  (write-unsigned-byte +tc-object+ stream)
  (write-class-desc (get-class-desc "java.lang.Integer") stream)
  (allocate-new-handle number)
  (write-signed-int number stream))

(defun write-java-long (number stream)
  (write-unsigned-byte +tc-object+ stream)
  (write-class-desc (get-class-desc "java.lang.Long") stream)
  (allocate-new-handle number)
  (write-signed-long number stream))

(defun write-java-float (number stream)
  (write-unsigned-byte +tc-object+ stream)
  (write-class-desc (get-class-desc "java.lang.Float") stream)
  (allocate-new-handle number)
  (write-float number stream))

(defun write-java-hashtable (hashtable stream)
  (write-unsigned-byte +tc-object+ stream)
  (write-class-desc (get-class-desc "java.util.Hashtable") stream)
  (allocate-new-handle hashtable)
  ; Default load factor.
  (write-float 0.75 stream)
  (let ((size (hash-table-count hashtable)))
    ; Is this okay for the threshold?
    (write-signed-int size stream)
    (let ((blockdata
           (with-output-to-vector (bd-stream)
             ; capacity
             (write-signed-int size bd-stream)
             ; size
             (write-signed-int size bd-stream))))
      ; objectAnnotation
      (write-blockdata blockdata stream)
      ; Key-value pairs
      (loop for key being the hash-key of hashtable
            for value being the hash-value of hashtable
            do (write-object key stream)
            (write-object value stream))
```

```
      (write-unsigned-byte +tc-endblockdata+ stream))))

(defun read-java-hashtable-classdata (stream)
  (let* ((load-factor (read-float stream))
         (threshold (read-signed-int stream))
         (annotation (read-object-annotation stream))
         hash-table)
    (debug-serialization "loadFactor: ~F" load-factor)
    (debug-serialization "threshold: ~D" threshold)
    load-factor threshold  ; ignore
    (with-input-from-vector (bd-stream (car annotation))
      (let ((capacity (read-signed-int bd-stream))
            (size (read-signed-int bd-stream)))
        (setq hash-table (make-hash-table :test #'equal))
        (debug-serialization "capacity: ~D" capacity)
        (debug-serialization "size: ~D" size)
        capacity  ; ignore
        size))
    (loop for key in (cdr annotation) by #'cddr
          for value in (cddr annotation) by #'cddr
          do (setf (gethash key hash-table) value))
    hash-table))

(defun write-java-vector (cons stream)
  (write-unsigned-byte +tc-object+ stream)
  (write-class-desc (get-class-desc "java.util.Vector") stream)
  (allocate-new-handle cons)
  ; default capacityIncrement
  (write-signed-int 0 stream)
  (let ((length (cons-length cons)))
    ; elementCount
    (write-signed-int length stream)
    ; elementData
    (write-new-array cons stream :length length)))

(defun read-java-vector-classdata (stream)
  (let* ((capacity-increment (read-signed-int stream))
         (element-count (read-signed-int stream))
         (element-data (read-object stream))
         (cons (make-list (1- element-count))))
    (debug-serialization "capacityIncrement: ~D" capacity-increment)
    capacity-increment  ; ignorable
    (debug-serialization "elementCount: ~D" element-count)
    (debug-serialization "elementData length = ~D" (length element-data))
    (loop for pair on cons
          for element across element-data
          do (setf (car pair) element))
    (setf (nthcdr (1- element-count) cons) (elt element-data (1- element-count)))
    cons))

(defun write-java-keyword (keyword stream)
  (write-unsigned-byte +tc-object+ stream)
  (write-class-desc (get-class-desc "lisp.lang.Keyword") stream)
  (allocate-new-handle keyword)
  (write-object (symbol-name keyword) stream))

(defun read-java-keyword-classdata (stream)
  (intern-keyword (read-object stream)))

(defun write-java-symbol (symbol stream)
  (write-unsigned-byte +tc-object+ stream)
  (write-class-desc (get-class-desc "lisp.lang.Symbol") stream)
  (allocate-new-handle symbol)
  (write-object (symbol-name symbol) stream)
  (write-object (short-package-name (symbol-package symbol)) stream))

(defun read-java-symbol-classdata (stream)
  (let ((name (read-object stream))
        (package (read-object stream)))
```

```
    (intern name package)))

(defun define-known-java-classes ()

  (define-java-class "java.lang.Object"
    :serial-version-uid #x-6F31A760EF8CD694)

  ; An array of type Object.
  (define-java-class "[Ljava.lang.Object;"
    :serial-version-uid #x-6F31A760EF8CD694)

  (define-java-class "java.lang.Boolean"
    :serial-version-uid #x-32DF8D7F2A630512
    :fields '((,+prim-typecode-boolean+ "value"))
    :object-writer #'write-java-boolean
    :classdata-reader #'read-java-boolean-classdata)

  (define-java-class "java.lang.Character"
    :serial-version-uid #x348B47D96B1A2678
    :fields '((,+prim-typecode-char+ "value"))
    :object-writer #'write-java-character
    :classdata-reader #'read-java-character-classdata)

  (define-java-class "java.lang.Number"
    :serial-version-uid #x-79536AE2F46B1F75)

  (define-java-class "java.lang.Byte"
    :super-class "java.lang.Number"
    :serial-version-uid #x-63B19F7B11AF0AE4
    :fields '((,+prim-typecode-byte+ "value"))
    :object-writer #'write-java-byte
    :classdata-reader #'read-signed-byte)

  (define-java-class "java.lang.Short"
    :super-class "java.lang.Number"
    :serial-version-uid #x684D37133460DA52
    :fields '((,+prim-typecode-short+ "value"))
    :object-writer #'write-java-short
    :classdata-reader #'read-signed-short)

  (define-java-class "java.lang.Integer"
    :super-class "java.lang.Number"
    :serial-version-uid #x12E2A0A4F7818738
    :fields '((,+prim-typecode-integer+ "value"))
    :object-writer #'write-java-integer
    :classdata-reader #'read-signed-int)

  (define-java-class "java.lang.Long"
    :super-class "java.lang.Number"
    :serial-version-uid #x3B8BE490CC8F23DF
    :fields '((,+prim-typecode-long+ "value"))
    :object-writer #'write-java-long
    :classdata-reader #'read-signed-long)

  (define-java-class "java.lang.Float"
    :super-class "java.lang.Number"
    :serial-version-uid #x-2512365D24C30F14
    :fields '((,+prim-typecode-float+ "value"))
    :object-writer #'write-java-float
    :classdata-reader #'read-float)

  (define-java-class "java.util.Hashtable"
    :serial-version-uid #x13BB0F25214AE4B8
    :class-desc-flags (logior +sc-write-method+ +sc-serializable+)
    :fields '((,+prim-typecode-float+ "loadFactor")
              (,+prim-typecode-integer+ "threshold"))
    :object-writer #'write-java-hashtable
    :classdata-reader #'read-java-hashtable-classdata)
```

```
  (define-java-class "java.util.Vector"
    :serial-version-uid #x-266882A47FC450FF
    :fields '((,+prim-typecode-integer+ "capacityIncrement")
              (,+prim-typecode-integer+ "elementCount")
              (,+obj-typecode-array+ "elementData" "[Ljava/lang/Object;"))
    :object-writer #'write-java-vector
    :classdata-reader #'read-java-vector-classdata)

  (define-java-class "lisp.lang.Keyword"
    :serial-version-uid #x1
    :fields '((,+obj-typecode-object+ "name" "Ljava/lang/String;"))
    :object-writer #'write-java-keyword
    :classdata-reader #'read-java-keyword-classdata)

  (define-java-class "lisp.lang.Symbol"
    :serial-version-uid #x1
    :fields '((,+obj-typecode-object+ "name" "Ljava/lang/String;")
              (,+obj-typecode-object+ "pkg" "Ljava/lang/String;"))
    :object-writer #'write-java-symbol
    :classdata-reader #'read-java-symbol-classdata))

; Register known Java classes.
(define-known-java-classes)

;;;-------------------------------------------------------------------
;;;
;;; SERIALIZATION ENVIRONMENTS
;;;

(defvar *reference-table* nil)

(defvar *next-handle* nil)

(defun allocate-new-handle (object)
  (setf (gethash object *reference-table*) *next-handle*
        (gethash *next-handle* *reference-table*) object)
  (prog1
    *next-handle*
    (incf *next-handle*)))

(defmacro with-new-handle-allocated ((handle-var) &body body)
  "Allocate a new handle, evaluate body, bind handle to result, return result."
  `(let* ((,handle-var (prog1 *next-handle* (incf *next-handle*)))
          (result (progn ,@body)))
     ,handle-var  ; ignorable
     (when result
       (setf (gethash result *reference-table*) ,handle-var
             (gethash ,handle-var *reference-table*) result)
       result)))

(defun get-handle (object)
  (gethash object *reference-table*))

(defun get-object-for-handle (handle)
  (gethash handle *reference-table*))

(defun clear-reference-table ()
  (clrhash *reference-table*))

(defun write-stream-preamble (stream)
  (write-unsigned-short +stream-magic+ stream)
  (write-unsigned-short +stream-version+ stream))

(defmacro with-java-serialization-output ((stream) &body body)
  `(let ((*next-handle* +base-wire-handle+)
         (*reference-table* (make-hash-table :test #'equal))
         (*bytes-written* 0)
         (*invocation-depth* 0))
     (declare (special *next-handle* *reference-table* *bytes-written* *invocation-depth*))
```

```
    (write-stream-preamble ,stream)
    ,@body))

(defun read-stream-preamble (stream)
  (cond ((= +stream-magic+ (read-unsigned-short stream))
         (debug-serialization "STREAM_MAGIC"))
        (t (error "STREAM_MAGIC not found on serialization input.")))
  (cond ((= +stream-version+ (read-unsigned-short stream))
         (debug-serialization "STREAM_VERSION = ~D" +stream-version+))
        (t (error "STREAM_VERSION not supported."))))

(defmacro with-java-serialization-input ((stream) &body body)
  `(let ((*next-handle* +base-wire-handle+)
         (*reference-table* (make-hash-table :test #'equal))
         (*bytes-read* 0))
     (declare (special *next-handle* *reference-table* *bytes-read*))
     (read-stream-preamble ,stream)
     ,@body))

;;;---------------------------------------------------------------------
;;;
;;; PROTOCOL IMPLEMENTATION
;;;

(defun write-null-reference (stream)
  (write-unsigned-byte +tc-null+ stream))

(defmethod write-new-array ((sequence vector) stream
                            &key (length (length sequence)))
  (write-unsigned-byte +tc-array+ stream)
  (write-class-desc (get-class-desc "[Ljava.lang.Object;") stream)
  (allocate-new-handle sequence)
  (write-signed-int length stream)
  (loop for thing across sequence
        do (write-object thing stream)))

(defmethod write-new-array ((sequence cons) stream
                            &key (length (cons-length sequence)))
  (write-unsigned-byte +tc-array+ stream)
  (write-class-desc (get-class-desc "[Ljava.lang.Object;") stream)
  (allocate-new-handle sequence)
  (write-signed-int length stream)
  (loop for p = sequence then (cdr p)
        while (consp p)
        do (write-object (car p) stream)
        finally (write-object p stream)))

(defun read-new-array (stream)
  (with-serialization-debugging ("TC_ARRAY")
    (read-class-desc stream)   ; ignore
    (let* ((size (read-unsigned-int stream))
           (array (make-array size))
           (handle (allocate-new-handle array)))
      (debug-serialization "newHandle: 0x~x" handle)
      handle ; ignorable
      (debug-serialization "size: ~D" size)
      (loop for idx from 0 upto (1- size)
            do (setf (elt array idx) (read-object stream)))
      (debug-serialization "values: ~S" array)
      array)))

(defun write-new-string (string stream)
  (write-unsigned-byte +tc-string+ stream)
  (allocate-new-handle string)
  (write-utf-string string stream))

(defun read-new-string (stream)
  (with-serialization-debugging ("TC_STRING")
    (let* ((string (read-utf-string stream))
```

```
          (handle (allocate-new-handle string)))
      (debug-serialization "newHandle: 0x~x" handle)
      handle ; ignorable
      (debug-serialization string)
      string)))

(defun write-prev-object (handle stream)
  (write-unsigned-byte +tc-reference+ stream)
  (write-signed-int handle stream))

(defun read-prev-object (stream)
  (with-serialization-debugging ("TC_REFERENCE")
    (let ((handle (read-signed-int stream)))
      (debug-serialization "handle: 0x~x" handle)
      (or (get-object-for-handle handle)
          (error "Could not find object with handle 0x~x." handle)))))

(defun write-field-desc (field-desc stream)
  (write-unsigned-byte (first field-desc) stream)
  (write-utf-string (second field-desc) stream)
  (when-bind (class-name1 (third field-desc))
    (write-new-string class-name1 stream)))

(defun read-field-desc (stream)
  (with-serialization-debugging ("fieldDesc")
    (let* ((typecode (read-unsigned-byte stream))
           (field-name (read-utf-string stream))
           (class-name1 (when (member typecode '(#.+obj-typecode-object+ #.+obj-typecode-a\
rray+))
                          (read-object stream))))
      (debug-serialization "~A: ~A" (if class-name1 "obj_typecode" "prim_typecode") (chara\
cter typecode))
      typecode  ; ignorable
      (debug-serialization "fieldName: ~A" field-name)
      field-name  ; ignorable
      (when class-name1
        (debug-serialization "className1: ~A" class-name1)))))

(defun write-fields (fields stream)
  (loop for field-desc in fields
        do (write-field-desc field-desc stream)))

(defun read-fields (stream)
  (with-serialization-debugging ("fields")
    (let ((count (read-unsigned-short stream)))
      (debug-serialization "count = ~D" count)
      (loop repeat count
            do (read-field-desc stream)))))

(defun write-class-annotation (annotation stream)
  (when annotation
    (loop for content in annotation
          do (write-object content stream)))
  (write-unsigned-byte +tc-endblockdata+ stream))

(defun read-class-annotation (stream)
  (with-serialization-debugging ("classAnnotation")
    (let ((annotation
           (loop for tc = (read-unsigned-byte stream)
                 until (= tc +tc-endblockdata+)
                 collect (read-object stream :typecode tc))))
      annotation)))

(defun write-object-annotation (annotation stream)
  (when annotation
    (loop for content in annotation
          do (write-object content stream)))
  (write-unsigned-byte +tc-endblockdata+ stream))
```

```
(defun read-object-annotation (stream)
  (with-serialization-debugging ("objectAnnotation")
    (let ((annotation
           (loop for tc = (read-unsigned-byte stream)
                 until (= tc +tc-endblockdata+)
                 collect (read-object stream :typecode tc))))
      annotation)))

(defun write-class-desc-info (class-desc stream)
  (with-slots (flags fields class-annotation super-class) class-desc
    (write-unsigned-byte flags stream)
    (write-unsigned-short (length fields) stream)
    (write-fields fields stream)
    (write-class-annotation class-annotation stream)
    (cond (super-class (write-class-desc super-class stream))
          (t (write-null-reference stream)))))

(defun read-class-desc-info (stream)
  (with-serialization-debugging ("classDescInfo")
    (let ((flags (read-unsigned-byte stream))
          (fields (read-fields stream))
          (annotation (read-class-annotation stream))
          (super-class (read-class-desc stream)))
      fields  ; ignore
      (debug-serialization "classDescFlags: ~D" flags)
      flags  ; ignorable
      (debug-serialization "classAnnotation: ~S" annotation)
      annotation  ; ignorable
      (debug-serialization "superClassDesc: ~S" super-class)
      super-class  ; ignorable
      )))

(defun write-new-class-desc (class-desc stream)
  (write-unsigned-byte +tc-classdesc+ stream)
  (with-slots (name serial-version-uid) class-desc
    (write-utf-string name stream)
    (write-signed-long serial-version-uid stream)
    (allocate-new-handle class-desc)
    (write-class-desc-info class-desc stream)))

(defun read-new-class-desc (stream)
  (with-serialization-debugging ("TC_CLASSDESC")
    (let* ((class-name (read-utf-string stream))
           (serial-version-uid (read-signed-long stream))
           (class-desc (get-class-desc class-name))
           (handle (allocate-new-handle class-desc)))
      (debug-serialization "className: ~A" class-name)
      (debug-serialization "serialVersionUID: 0x~x" serial-version-uid)
      serial-version-uid  ; ignorable
      (debug-serialization "newHandle: 0x~x" handle)
      handle  ; ignorable
      (read-class-desc-info stream)
      class-desc)))

(defun write-class-desc (class-desc stream)
  (let ((handle (get-handle class-desc)))
    (cond (handle (write-prev-object handle stream))
          (t (write-new-class-desc class-desc stream)))))

(defun read-class-desc (stream)
  (with-serialization-debugging ("classDesc")
    (let ((tc (read-unsigned-byte stream)))
      (ecase tc
        (#.+tc-classdesc+ (read-new-class-desc stream))
        (#.+tc-null+ (debug-serialization "TC_NULL") nil)
        (#.+tc-reference+ (read-prev-object stream))))))

(defun write-classdata (class-desc instance stream)
  (with-slots (flags slot-names) class-desc
```

```
    (unless (= +sc-serializable+ flags)
      (error "Implement other classdata parsing modes."))
    (loop for slot in slot-names
          do (write-object (slot-value instance slot) stream))))

(defmethod read-classdata ((class-desc java-class-desc) stream)
  (with-new-handle-allocated (handle)
    (debug-serialization "newHandle: 0x~x" handle)
    (with-serialization-debugging ("classdata")
      (funcall (class-desc-classdata-reader class-desc) stream))))

(defmethod read-classdata ((class-desc lisp-class-desc) stream)
  (with-serialization-debugging ("classdata")
    (with-slots (class-name slot-names) class-desc
      (let ((instance (make-instance class-name)))
        (allocate-new-handle instance)
        (loop for slot in slot-names
              do (setf (slot-value instance slot) (read-object stream)))
        instance))))

(defun write-new-object (object stream)
  (let ((class-desc (get-class-desc (class-of object))))
    (write-unsigned-byte +tc-object+ stream)
    (write-class-desc class-desc stream)
    (allocate-new-handle object)
    (write-classdata class-desc object stream)))

(defun read-new-object (stream)
  (with-serialization-debugging ("TC_OBJECT")
    (let ((object (read-classdata (read-class-desc stream) stream)))
      (debug-serialization "classdata: ~S" object)
      object)))

(defun write-new-class (class stream)
  (write-unsigned-byte +tc-class+ stream)
  (let ((class-desc (get-class-desc class)))
    (write-class-desc class-desc stream)
    (allocate-new-handle class-desc)))

(defun read-new-class (stream)
  (with-serialization-debugging ("TC_CLASS")
    (let ((class-desc (read-class-desc stream)))
      (allocate-new-handle class-desc))))

(defun write-blockdata (vector stream)
  (let ((size (length vector)))
    (cond ((< size #x100)
           (write-unsigned-byte +tc-blockdata+ stream)
           (write-unsigned-byte size stream))
          ((< size #x80000000)
           (write-unsigned-byte +tc-blockdatalong+ stream)
           (write-signed-int size stream))
          (t (error "Vector too long to be written as blockdata.")))
    (write-signed-bytes vector stream)))

(defun read-blockdata-short (stream)
  (with-serialization-debugging ("blockdatashort")
    (let ((size (read-unsigned-byte stream)))
      (debug-serialization "size: ~D" size)
      (with-output-to-vector (bd-stream)
        (loop repeat size
              do (write-unsigned-byte (read-unsigned-byte stream) bd-stream))))))

(defun read-blockdata-long (stream)
  (with-serialization-debugging ("blockdatalong")
    (let ((size (read-unsigned-byte stream)))
      (debug-serialization "size: ~D" size)
      (with-output-to-vector (bd-stream)
        (loop repeat size
```

```
            do (write-unsigned-byte (read-unsigned-byte stream) bd-stream))))))

(defun read-exception (stream)
  (declare (ignore stream))
  (with-serialization-debugging ("TC_EXCEPTION")
    (clear-reference-table)
    (error "Implement reading (Throwable) object.")
    (clear-reference-table)))

(defgeneric %write-object (thing stream)
  (:documentation "Serialize something."))

(defmethod %write-object (thing stream)
  (write-new-object thing stream))

(defmethod %write-object ((thing (eql t)) stream)
  (write-java-boolean t stream))

(defmethod %write-object ((thing character) stream)
  (write-java-character thing stream))

(defmethod %write-object ((thing class) stream)
  (write-new-class thing stream))

(defmethod %write-object ((thing null) stream)
  (write-null-reference stream))

(defmethod %write-object ((thing sequence) stream)
  (write-new-array thing stream))

(defmethod %write-object ((thing cons) stream)
  (write-java-vector thing stream))

(defmethod %write-object ((thing string) stream)
  (write-new-string thing stream))

(defmethod %write-object ((thing hash-table) stream)
  (write-java-hashtable thing stream))

(defmethod %write-object ((thing symbol) stream)
  (write-java-symbol thing stream))

(defmethod %write-object ((thing keyword) stream)
  (write-java-keyword thing stream))

(defmethod %write-object ((thing integer) stream)
  (let ((byte-length (ceiling (/ (1+ (integer-length thing)) 8))))
    (cond ((<= byte-length 1) (write-java-byte thing stream))
          ((<= byte-length 2) (write-java-short thing stream))
          ((<= byte-length 4) (write-java-integer thing stream))
          ((<= byte-length 8) (write-java-long thing stream))
          (t (error "Integer translation not implemented.")))))

; Decide here whether a number should be written as float or double.
(defmethod %write-object ((thing float) stream)
  (write-java-float thing stream))

(defvar *invocation-depth* 0
  "Used to determine if a call to write-object occurs while writing another object.")

;;;-------------------------------------------------------------------
;;;
;;; SERIALIZATION API
;;;

(defun flush-cached-class-mappings ()
  "Decache precomputed mappings between CLOS and Java classes."
  (clrhash *class-desc-table*)
  (define-known-java-classes))
```

```
(defun output-byte-count ()
  *bytes-written*)

(defun input-byte-count ()
  *bytes-read*)

(defgeneric instance-for-serialization (instance depth)
  (:documentation "Classes may override this function to provide a substitute instance
    at serialization time, e.g. a network identifier instead of the actual data."))

(defmethod instance-for-serialization (instance depth)
  (declare (ignore depth))
  instance)

(defun write-object (thing stream)
  (let* ((actual-thing (instance-for-serialization thing *invocation-depth*))
         (handle (when actual-thing (get-handle actual-thing)))
         (*invocation-depth* (1+ *invocation-depth*)))
    (declare (special *invocation-depth*))
    (cond (handle (write-prev-object handle stream))
          (t (%write-object actual-thing stream)))))

(defun read-object (stream &key eof-errorp eof-value typecode)
  (let ((tc (or typecode (read-unsigned-byte stream))))
    (cond (tc (ecase tc
                (#.+tc-object+ (read-new-object stream))
                (#.+tc-blockdata+ (read-blockdata-short stream))
                (#.+tc-blockdatalong+ (read-blockdata-long stream))
                (#.+tc-class+ (read-new-class stream))
                (#.+tc-array+ (read-new-array stream))
                (#.+tc-string+ (read-new-string stream))
                (#.+tc-classdesc+ (read-new-class-desc stream))
                (#.+tc-reference+ (read-prev-object stream))
                (#.+tc-null+ (debug-serialization "TC_NULL") nil)
                (#.+tc-exception+ (read-exception stream))
                (#.+tc-reset+ (debug-serialization "TC_RESET") (clear-reference-table))))
          (eof-errorp (error "EOF reached while reading from serialization stream."))
          (t eof-value))))

;; Handle long class names.
(defun generate-java-source-for-class (class-name directory)
  (let* ((class (find-class class-name :error-p t))
         (class-desc (get-class-desc class))
         (java-class-name (class-desc-name class-desc))
         (name-start (1+ (position #\. java-class-name :from-end t)))
         (short-class-name (subseq java-class-name name-start))
         (file-name (concatenate 'string short-class-name ".java"))
         (pathname (merge-pathnames file-name directory))
         (fields (class-desc-fields class-desc)))
    (with-open-file (stream pathname
                            :direction :output
                            :if-exists :supersede
                            :if-does-not-exist :create)
      (format stream "// This is a generated file.
// Mapping for Common Lisp class ~A.

package lisp.map;

import java.io.Serializable;
import lisp.lang.*;

public class ~A extends LispObject implements Serializable {

private static final long serialVersionUID = 1;

"
              (let ((*package* *empty-package*))
                (write-to-string class-name))
```

```
            short-class-name)
      (loop for field in fields
            do (format stream "public Object ~A;~%" (second field)))
      (format stream "~%public ~A () {}~%~%}" short-class-name)
    pathname)))

(defun generate-java-sources (specs directory &key verbose)
  (loop for class-name in specs
        for pathname = (generate-java-source-for-class class-name directory)
        do (when verbose
             (format t "~%; Generated ~S" pathname))))

;;;---------------------------------------------------------------------
;;;
;;; EXAMPLE
;;;

#||

(defclass foo () ((x :initarg :x)))

(defvar *foo* (make-instance 'foo :x '(1 2 3)))

(with-open-file (stream (pathname "cds:code;java;test-dump")
                        :direction :output
                        :if-exists :supersede
                        :if-does-not-exist :create
                        :element-type '(unsigned-byte 8))
  (with-java-serialization-output (stream)
    (write-object (make-instance 'foo :x `("This is a string."
                                           7
                                           12345678987654321
                                           -512
                                           (0 1 2 3)
                                           ,(intern "FOO-BAR" "CL-USER")))
                  stream)))

(with-open-file (stream (pathname "cds:code;java;test-dump")
                        :direction :input
                        :element-type '(unsigned-byte 8))
  (with-java-serialization-input (stream)
    (read-object stream)))

(generate-java-sources '(foo) (pathname "cds:code;java;lisp;map;") :verbose t)

; Compile Jos_Foo.java and TestSerial.java, run TestSerial.

;------------------------ TestSerial.java -----------------------

import java.io.*;
import lisp.lang.*;
import lisp.map.*;

public class TestSerial {

public TestSerial() {
  try {
    FileInputStream in = new FileInputStream("test-dump");
    ObjectInputStream s2 = new ObjectInputStream(in);
    Jos_Foo a = (Jos_Foo)s2.readObject();
    Object[] x = (Object[])a.x;
    for(int i = 0; i < x.length; i++) {
       System.out.println(x[i]);
    }
  } catch (Exception e) { e.printStackTrace(); }
}

public static void main(String[] args) {
  TestSerial app = new TestSerial();
```

```
}

}

||#
```

11.3 Persistent Object Support

11.3.1 code/pcs/bin-dumper.lisp

```
;;; -*- Package: PCS; Mode: LISP; Syntax: Common-lisp; Base: 10 -*-

;;; (C) Copyright 1997, Rusty Johnson and Andrew J. Blumberg (blumberg@ai.mit.edu)
;;;     All Rights Reserved.
;;;
;;;     Thanks to Steve Hain (slh@digitoo.com) for various fixes and improvements.
;;;
;;;     Adapted and extended for PCS by Christopher Vincent (cvince@mit.edu) 1/19/99.

;;;; This file contains a CUSTOM binary dumper and loader technology that has
;;;; been tuned for dumping and loading database tables. These tables should
;;;; be readable on any machine / Lisp implementation with some adjustments to
;;;; this file.

;;;; Binary encoding scheme

;;;; If the high order bit is a 0,  this 16-bit word is a positive 15-bit
;;;; integer (0 <= n < 1_15)
;;;;
;;;; If the high order bits are #b10,  then this word contains the 14 high
;;;; order bits of a positive 30-bit integer  (0 <= n < 1_30), the 16
;;;; low order bits are in the next word.
;;;;
;;;; If the this order bits are #b11,  then the following 14 bits are the
;;;; extended tag as follows:
;;;;
;;;;    TAG                Meaning
;;;;  #b0xxxxxxxxxxxxxxx  positive 15-bit integer (0 <= n < 1_15)
;;;;  #b10xxxxxxxxxxxxxx  positive 30-bit integer (0 <= n < 1_30), low order bits in next \
word.
;;;;  #b1100000000000000  Boolean False
;;;;  #b1111111111111111  Boolean True
;;;;  #b1100000000000001  32 bit fixnum
;;;;  #b1100000000000100  string
;;;;  #b1100000000000101  symbol
;;;;  #b1100000000000111  list
;;;;  #b1100000000001000  vise-server:index-table
;;;;  #b1100000000001001  vise-server:reversible-index-table
;;;;  #b1100000000001010  array
;;;;  #b1100000000001100  improper list
;;;;  #b1100110011001100  other


;;;; Should be:
;;;;  #b0xxxxxxxxxxxxxxx  positive 15-bit integer (0 <= n < 1_15)
;;;;  #b10xxxxxxxxxxxxx  positive 14-bit increment from the previous integer
;;;;  #b110xxxxxxxxxxxx  next N (13-bit) entries are increments from the previous integer
;;;;  #b1110xxxxxxxxxxx  positive 27-bit integer (0 <= n < 1_27), low order bits in next \
word.
;;;;  #b1111000000000000  Boolean False
;;;;  #b1111111111111111  Boolean True
;;;;  #b1111000000000001  32 bit fixnum
;;;;  #b1111000000000010  ascending list of integers
;;;;  #b1111000000000100  string
```

```
;;;;  #b1111000000000101  symbol
;;;;  #b1111000000000111  list
;;;;  #b1111000000001000  vise-server:index-table
;;;;  #b1111000000001001  vise-server:reversible-index-table
;;;;  #b1111000000001010  array
;;;;  #b1111000000001100  improper list
;;;;  #b1111110011001100  other

(in-package :pcs )

;;;-------------------------------------------------------------------
;;;
;;; CONSTANTS
;;;

(eval-when (:compile-toplevel :load-toplevel :execute)
  (defconstant +false-tag+                        #b11100000)
  (defconstant +true-tag+                         #b11111110)
  (defconstant +32-bit-fixnum-tag+                #b11100001)
  (defconstant +32-bit-float-tag+                 #b11100010)
  (defconstant +64-bit-float-tag+                 #b11100011)
  (defconstant +string-tag+                       #b11100100)
  (defconstant +symbol-tag+                       #b11100101)
  (defconstant +list-tag+                         #b11100111)
  (defconstant +pcs-identifier-tag+               #b11101000)
  (defconstant +pcs-instance-tag+                 #b11101001)
  (defconstant +pcs-slot-unbound-tag+             #b11101101)
  (defconstant +array-tag+                        #b11101010)
  (defconstant +improper-list-tag+                #b11101100)
  (defconstant +hash-table-tag+                   #b11101110)
  (defconstant +pcs-instance-stub-tag+            #b11101111)
  (defconstant +bignum-tag+                       #b11110000)
  ;;(defconstant +unassigned+                     #b11110001)
  ;;(defconstant +unassigned+                     #b11110010)
  ;;(defconstant +unassigned+                     #b11110011)
  (defconstant +negative-tag+                     #b11110100)
  (defconstant +other-tag+                        #b11111100)
  (defconstant +end-tag+                          #b11111111)
  (defconstant +byte-8-0+  (byte 8  0))
  (defconstant +byte-8-8+ (byte 8 8))
  (defconstant +byte-8-16+ (byte 8 16))
  (defconstant +byte-8-24+ (byte 8 24))
  (defconstant +byte-5-16+ (byte 5 16))
  (defconstant +byte-3-5+ (byte 3 5))
  (defconstant +byte-6-8+ (byte 6 8))
  (defconstant +byte-6-0+ (byte 6 0))
  (defconstant +byte-5-0+ (byte 5 0))

  (defconstant +2-byte-fixnum-base+ #b10000000)
  (defconstant +3-byte-fixnum-base+ #b11000000)

  (defconstant +byte-fixnum+ (ash 1 7))
  (defconstant +2-byte-fixnum+ (ash 1 14))
  (defconstant +3-byte-fixnum+ (ash 1 21))

  (defconstant +byte-1-7+ (byte 1 7))
  (defconstant +byte-1-6+ (byte 1 6))
  (defconstant +byte-1-5+ (byte 1 5))

  (defconstant +ash-1-31+   (ash 1 31))
  (defconstant +ash-1-32+   (ash 1 32)))

;;;-------------------------------------------------------------------
;;;
;;; PARAMETERS
;;;

(defparameter *op-code-table* nil)
(defparameter *op-code-skip-table* nil)
```

```
(defparameter *version* 0)
(defparameter *default-hash-table-test* #'equalp)

(defparameter *2-power-array* #(1 2 4 8 16 32 64 128))

;;;------------------------------------------------------------------
;;;
;;; PLATFORM SPECIFIC
;;;

#+MCL
(defparameter *buffer* (make-array 0 :adjustable t :fill-pointer t
   :element-type 'base-character))
#-MCL
(defparameter *buffer* (make-array 0 :adjustable t :fill-pointer t))

#+MCL
(eval-when (:execute :compile-toplevel)
  (shadow 'write-byte :pcs))

#+MCL
(eval-when (:execute :compile-toplevel)
  (defmacro write-byte (byte stream)
    `(ccl::stream-write-byte ,stream ,byte)))

;; When in Genera, scl:pkg-find-package gives better error recovery.
#+Genera
(defmacro %find-package (package)
  `(scl:pkg-find-package ,package))
#-Genera
(defmacro %find-package (package)
  `(find-package ,package))

#+Genera
(defmacro %read-next-byte (stream)
  `(si:bin-next-byte ,stream))

#+MCL
(defmacro %read-next-byte (stream)
  `(ccl::stream-read-byte ,stream))

(defmacro %clear-next-byte (stream)
  `(%read-next-byte ,stream))

(defmacro %clear-bytes (stream n)
  `(loop repeat ,n
         do (%read-next-byte ,stream)))

#-(OR Genera MCL)
(defmacro %read-next-byte (stream)
  `(read-byte ,stream))

#+(OR Genera MCL)
(declaim (inline logdpb))
#+Genera
(defun logdpb (new-byte byte-spec integer)
  (sys:%logdpb new-byte byte-spec integer))

#-Genera
(defun logdpb (new-byte byte-spec integer)
  (let ((temp (dpb new-byte byte-spec integer)))
    (when (> temp +ash-1-31+)
      (setq temp (- temp +ash-1-32+)))
    temp))

#+Genera
(defmacro string-out (vector stream)
  `(compiler:string-out ,vector ,stream))
```

```
#-Genera
(defmacro string-out (vector stream)
  `(loop for item across ,vector
do (write-byte item ,stream)))

#+Genera
(defmacro string-in (string stream)
  `(flavor::send ,stream ':string-in nil ,string))

#-Genera
(defmacro string-in (string stream)
  `(loop for idx #+MCL fixnum from 0 below (length ,string)
 do (setf (aref ,string idx) (code-char (read-byte ,stream)))))

#+Genera
(defmacro bit-vector-out (vector stream &optional length)
  `(let* ((vector-length (or ,length (length ,vector)))
  (stack-array-length (floor vector-length 8))
  (slop (* stack-array-length 8)))
     (sys:with-stack-array (n-vector stack-array-length :displaced-to ,vector :element-typ\
e '(unsigned-byte 8))
       (string-out n-vector ,stream))
     (write-byte (loop for idx upfrom slop below vector-length
       for bit = (aref ,vector idx)
       for entry across *2-power-array*
       sum (* bit entry))
 stream)))

#-Genera
(defmacro bit-vector-out (vector stream &optional length)
  `(loop for idx #+MCL fixnum from 0 below (or ,length (length ,vector))
 do (write-byte (aref ,vector idx) ,stream)))

#+Genera
(defmacro bit-vector-in (vector stream &optional length)
  `(let* ((vector-length (or ,length (length ,vector)))
  (stack-array-length (floor vector-length 8))
  (slop (* stack-array-length 8)))
     (sys:with-stack-array (n-vector (floor (or ,length (length ,vector)) 8) :displaced-to\
 ,vector :element-type '(unsigned-byte 8))
       (string-in n-vector ,stream))
     (loop with byte = (%read-next-byte stream)
   for index upfrom slop below vector-length
   for idx upfrom 0
   for bit = (ldb (byte 1 idx) byte)
   do (setf (aref ,vector index) bit))))

#-Genera
(defmacro bit-vector-in (vector stream &optional length)
  `(loop for idx #+MCL fixnum from 0 below (or ,length (length ,vector))
 do (setf (aref ,vector idx) (read-byte ,stream))))

#+Genera
(defun read-string (stream)
  (let* ((length (read-thing stream))
 (string (make-array length :element-type 'lisp:string-char))
 (displaced-string (make-array length :displaced-to string :element-type '(unsigned-byte \
8))))
    (string-in displaced-string stream)
    string))

#-Genera
(defun read-string (stream)
  (let* ((length (read-thing stream))
 (string (make-array length :element-type http::*standard-character-type*)))
    (string-in string stream)
    string))

(defun skip-string (stream)
```

```
  (%clear-bytes stream (read-thing stream)))

;;;---------------------------------------------------------------------
;;;
;;; CODE
;;;

(declaim (inline write-32-bit-fixnum
 read-32-bit-fixnum
 write-21-bit-fixnum
 write-14-bit-fixnum
 write-7-bit-fixnum
 dump-negative))

(defun write-32-bit-fixnum-data (fixnum stream)
  (write-byte (ldb +byte-8-24+ fixnum) stream)
  (write-byte (ldb +byte-8-16+  fixnum) stream)
  (write-byte (ldb +byte-8-8+  fixnum) stream)
  (write-byte (ldb +byte-8-0+  fixnum) stream))

(defun read-32-bit-fixnum-data (stream)
  (logdpb (%read-next-byte stream) +byte-8-24+
  (logdpb (%read-next-byte stream) +byte-8-16+
  (logdpb (%read-next-byte stream) +byte-8-8+
  (%read-next-byte stream)))))

(defun skip-32-bit-fixnum-data (stream)
  (%clear-bytes stream 4))

(defun write-21-bit-fixnum-data (fixnum stream)
  (write-byte (logdpb (ldb +byte-5-16+ fixnum) +byte-5-0+ +3-byte-fixnum-base+) stream)
  (write-byte (ldb +byte-8-8+ fixnum) stream)
  (write-byte (ldb +byte-8-0+ fixnum) stream))

(defun write-14-bit-fixnum-data (fixnum stream)
  (write-byte (logdpb (ldb +byte-6-8+ fixnum) +byte-6-0+ +2-byte-fixnum-base+) stream)
  (write-byte (ldb +byte-8-0+ fixnum) stream))

(defun write-7-bit-fixnum-data (fixnum stream)
  (write-byte fixnum stream))

;;;; fixnums
(defun read-32-bit-fixnum (stream)
  (read-32-bit-fixnum-data stream))

(defun skip-32-bit-fixnum (stream)
  (skip-32-bit-fixnum-data stream))

(defun dump-negative (fixnum stream)
  (write-byte +negative-tag+ stream)
  (dump-fixnum (- fixnum) stream))

(defun read-negative (stream)
  (- (read-thing stream)))

(defun skip-negative (stream)
  (skip-thing stream))

(defun dump-fixnum (fixnum stream)
  (cond ((< fixnum 0)
 (dump-negative fixnum stream))
((< fixnum +byte-fixnum+)
 (write-7-bit-fixnum-data fixnum stream))
((< fixnum +2-byte-fixnum+)
 (write-14-bit-fixnum-data fixnum stream))
((< fixnum +3-byte-fixnum+)
 (write-21-bit-fixnum-data fixnum stream))
(t (write-byte +32-bit-fixnum-tag+ stream)
   (write-32-bit-fixnum-data fixnum stream))))
```

```
;;;; bignums

(defun dump-bignum (bignum stream)
  (when (< bignum 0)
    (write-byte +negative-tag+ stream)
    (setq bignum (- bignum)))
  (write-byte +bignum-tag+ stream)
  (write-byte (1- (ceiling (/ (log (1+ bignum) 2) 8))) stream)
  (loop while (> bignum 0)
        do (write-byte (logand 255 bignum) stream)
           (setq bignum (ash bignum -8))))

(defun read-bignum (stream)
  (loop for idx upfrom 0 to (read-byte stream)
        with bignum = 0
        do (setq bignum (+ bignum (ash (read-byte stream) (* 8 idx))))
        finally (return bignum)))

(defun skip-bignum (stream)
  (let ((b (read-byte stream)))
    (when (= b +negative-tag+)
      (setq b (read-byte stream)))
    (%clear-bytes stream (1+ b))))

;;;; strings

;; do not be tempted to convert this to use ':string-out; fat-strings screw you hard if yo\
u try this.
;; 4/28/98 09:52:48 AJB

(defun dump-string (string stream)
  (write-byte +string-tag+ stream)
  (let* ((length (length string)))
    (dump-thing length stream)
    (loop for item across string
  do (write-byte (char-code item) stream))))

(defun list-length-blah (list)
  (do ((count 0 (1+ count))
       (p list (cdr p)))
      ((not (consp p)) (values count (null p)))))
;;;; lists (proper and improper)
(defun dump-list (list stream)
  (multiple-value-bind (length proper-p)
      (do ((count 0 (1+ count))
           (p list (cdr p)))
          ((not (consp p)) (values count (null p))))
    (write-byte (if proper-p +list-tag+ +improper-list-tag+) stream)
    (dump-fixnum length stream)
    (do ((p list (cdr p)))
        ((not (consp p))
         (when p
           (dump-thing p stream)))
      (dump-thing (car p) stream))))

(defun read-list (stream)
  (let ((new-list (make-list (read-thing stream))))
    (loop for temp on new-list
          do (setf (car temp) (read-thing stream)))
    new-list))

(defun skip-list (stream)
  (skip-things stream (read-thing stream)))

(defun read-improper-list (stream)
  (let ((new-list (make-list (read-thing stream))))
    (do ((p    new-list       next)
         (next (cdr new-list) (cdr next)))
```

```
        ((null next)
         (setf (car p) (read-thing stream)
               (cdr p) (read-thing stream)))
      (setf (car p) (read-thing stream)))
    new-list))

(defun skip-improper-list (stream)
  (skip-things stream (1+ (read-thing stream))))

;;;; ARRAYs
(defun dump-array (array stream)
  (write-byte +array-tag+ stream)
  (let ((length (length array)))
    (dump-fixnum length stream)
    (loop for thing across array
          do (dump-thing thing stream))))

(defun read-array (stream)
  (let* ((length (read-thing stream))
 (new-array (make-array length :adjustable t :fill-pointer t)))
    (loop for idx upfrom 0 below length
  do (setf (aref new-array idx) (read-thing stream)))
    new-array))

(defun skip-array (stream)
  (skip-things stream (read-thing stream)))

;;;; SYMBOLs

;; Trade time for space, abbreviate package names.
(defun dump-symbol (symbol stream)
  (write-byte +symbol-tag+ stream)
  (dump-thing (symbol-name symbol) stream)
  (dump-thing (short-package-name (symbol-package symbol)) stream))

(defun read-symbol (stream)
  (let ((name (read-thing stream))
        (pkg (read-thing stream)))
    (intern name (%find-package pkg))))

(defun skip-symbol (stream)
  (skip-things stream 2))

;;;; PCS

(defun dump-pcs-identifier (identifier stream)
  (write-byte +pcs-identifier-tag+ stream)
  (dump-thing (identifier-serial-number identifier) stream)
  (dump-thing (identifier-class-name identifier) stream)
  (dump-thing (identifier-storage-mechanism-name identifier) stream))

(defun read-pcs-identifier (stream)
  (allocate-identifier (read-thing stream)
                       (read-thing stream)
                       (read-thing stream)))

(defun skip-pcs-identifier (stream)
  (skip-things stream 3))

(defgeneric dump-instance-slot-values (instance stream)
  (:documentation "Dump slot values to a bin-dumper stream."))

;; Stubs are generated by default, this must be called directly to
;; save a complete instance.
(defun dump-pcs-instance (instance stream)
  (write-byte +pcs-instance-tag+ stream)
  (dump-thing (persistent-instance-identifier instance) stream)
  (dump-instance-slot-values instance stream))
```

```
(defgeneric read-instance-slot-values (instance stream)
  (:documentation "Fill appropriate slot values from a bin-dumper stream."))

(defgeneric skip-instance-slot-values (class-name stream)
  (:documentation "Skip appropriate slot values from a bin-dumper stream."))

(defun read-pcs-instance (stream)
  (read-instance-slot-values (allocate-instance-for-identifier (read-thing stream)) stream\
))

(defun skip-pcs-instance (stream)
  ;; Manually read identifier
  (%clear-next-byte stream) ; pcs-identifier tag
  (let ((serial-number (read-thing stream))
        (class-name (read-thing stream)))
    (check-type serial-number integer)
    (skip-thing stream) ; storage-mechanism name
    (skip-instance-slot-values class-name stream)
    serial-number))

(defun dump-pcs-instance-stub (instance stream)
  (write-byte +pcs-instance-stub-tag+ stream)
  (dump-thing (persistent-instance-identifier instance) stream))

(defun read-pcs-instance-stub (stream)
  (allocate-instance-for-identifier (read-thing stream)))

(defun skip-pcs-instance-stub (stream)
  (skip-thing stream))

(defun dump-pcs-slot-unbound (stream)
  "Special dump to denote an unbound slot."
  (write-byte +pcs-slot-unbound-tag+ stream))

(defclass unbound-slot-marker () ()
  (:documentation "Identifies an unbound slot."))

(defparameter *unbound-slot* (make-instance 'unbound-slot-marker))

(defgeneric unbound-slot-p (thing)
  (:documentation "Determine if an object is an unbound slot marker."))

(defmethod unbound-slot-p ((thing unbound-slot-marker))
  t)

(defmethod unbound-slot-p (thing)
  (declare (ignore thing))
  nil)

(defun read-pcs-slot-unbound (stream)
  "Return a special object denoting an unbound slot."
  (declare (ignore stream))
  *unbound-slot*)

(defun skip-pcs-slot-unbound (stream)
  (declare (ignore stream)))

(defun dump-hash-table (hash-table stream)
  (write-byte +hash-table-tag+ stream)
  (dump-thing (hash-table-rehash-size hash-table) stream)
  (dump-thing (hash-table-rehash-threshold hash-table) stream)
  (dump-thing (hash-table-size hash-table) stream)
  (dump-thing (hash-table-test hash-table) stream)
  (dump-thing (hash-table-count hash-table) stream)
  (loop for key being the hash-key of hash-table
        for value being the hash-value of hash-table
        do (dump-thing key stream)
           (dump-thing value stream)))
```

```
(defun read-hash-table (stream)
  (let ((table (make-hash-table
                :rehash-size (read-thing stream)
                :rehash-threshold (read-thing stream)
                :size (read-thing stream)
                :test (read-thing stream)))
        (count (read-thing stream)))
    (loop repeat count
          for key = (read-thing stream)
          for value = (read-thing stream)
          do (setf (gethash key table) value))
    table))

(defun skip-hash-table (stream)
  (skip-things stream 4)
  (skip-things stream (* 2 (read-thing stream))))

;; other, Lisp readable

(defun dump-other (thing stream)
  (write-byte +other-tag+ stream)
  (setf (fill-pointer *buffer*) 0)
  (with-output-to-string (s1 *buffer*)
    (format s1 "~S" thing))
  (dump-string *buffer* stream))

(defun read-other (stream)
  (read-from-string (read-thing stream)))

(defun skip-other (stream)
  (skip-thing stream))

;;;; DUMP- & READ- THING

(defgeneric dump-thing (thing stream)
  (:documentation "Method dispatch to dump; specialize to extend."))

(defmethod dump-thing ((thing fixnum) stream)
  (dump-fixnum thing stream))

(defmethod dump-thing ((thing integer) stream)
  (dump-bignum thing stream))

(defmethod dump-thing ((thing string) stream)
  (dump-string thing stream))

(defmethod dump-thing ((thing (eql nil)) stream)
  (write-byte +false-tag+ stream))

(defmethod dump-thing ((thing (eql t)) stream)
  (write-byte +true-tag+ stream))

(defmethod dump-thing ((thing list) stream)
  (dump-list thing stream))

(defmethod dump-thing ((thing symbol) stream)
  (dump-symbol thing stream))

(defmethod dump-thing ((thing array) stream)
  (dump-array thing stream))

(defmethod dump-thing ((thing hash-table) stream)
  (dump-hash-table thing stream))

(defmethod dump-thing ((thing identifier) stream)
  (dump-pcs-identifier thing stream))

(defmethod dump-thing ((thing persistent-instance-mixin) stream)
  (dump-pcs-instance-stub thing stream))
```

```
(defmethod dump-thing (thing stream)
  (dump-other thing stream))

;;

(defun read-thing (stream)
  (let ((item (%read-next-byte stream))
reader)
    (cond ((not (ldb-test +byte-1-7+ item))
   item)
  ((not (ldb-test +byte-1-6+ item))
   (logdpb (ldb +byte-6-0+ item) +byte-8-8+ (%read-next-byte stream)))
  ((not (ldb-test +byte-1-5+ item))
   (logdpb (ldb +byte-5-0+ item) +byte-8-16+ (logdpb (%read-next-byte stream) +byte-8-8+ \
(%read-next-byte stream))))
  ((setq reader (aref *op-code-table* item))
   (funcall reader stream))
  (t (error "Unknown opcode ~S" item)))))

(defun skip-thing (stream)
  (let ((item (%read-next-byte stream))
reader)
    (cond ((not (ldb-test +byte-1-7+ item)) nil)
  ((not (ldb-test +byte-1-6+ item)) (%clear-next-byte stream) nil)
  ((not (ldb-test +byte-1-5+ item)) (%clear-bytes stream 2) nil)
  ((setq reader (aref *op-code-skip-table* item))
   (funcall reader stream))
  (t (error "Unknown opcode ~S" item)))))

(defun skip-things (stream n)
  (loop repeat n
        do (skip-thing stream)))

(defun initialize-op-code-table ()
  (let ((table (make-array 256 :initial-element nil)))
    (loop for (code . symbol) in '((#.+hash-table-tag+ . read-hash-table)
   (#.+32-bit-fixnum-tag+ . read-32-bit-fixnum)
                                  (#.+bignum-tag+ . read-bignum)
   (#.+string-tag+ . read-string)
   (#.+array-tag+ . read-array)
   (#.+list-tag+ . read-list)
   (#.+symbol-tag+ . read-symbol)
   (#.+negative-tag+ . read-negative)
   (#.+other-tag+ . read-other)
   (#.+improper-list-tag+ . read-improper-list)
                                  (#.+pcs-identifier-tag+ . read-pcs-identifier)
                                  (#.+pcs-instance-tag+ . read-pcs-instance)
                                  (#.+pcs-slot-unbound-tag+ . read-pcs-slot-unbound)
                                  (#.+pcs-instance-stub-tag+ . read-pcs-instance-stub))
  do (setf (aref table code) (symbol-function symbol)))
    (setf (aref table +true-tag+) #'(lambda (x) (declare (ignore x)) t))
    (setf (aref table +false-tag+) #'(lambda (x) (declare (ignore x)) nil))
    (setf *op-code-table* table)))

(defun initialize-op-code-skip-table ()
  (let ((table (make-array 256 :initial-element nil)))
    (loop for (code . symbol) in '((#.+hash-table-tag+ . skip-hash-table)
   (#.+32-bit-fixnum-tag+ . skip-32-bit-fixnum)
                                  (#.+bignum-tag+ . skip-bignum)
   (#.+string-tag+ . skip-string)
   (#.+array-tag+ . skip-array)
   (#.+list-tag+ . skip-list)
   (#.+symbol-tag+ . skip-symbol)
   (#.+negative-tag+ . skip-negative)
   (#.+other-tag+ . skip-other)
   (#.+improper-list-tag+ . skip-improper-list)
                                  (#.+pcs-identifier-tag+ . skip-pcs-identifier)
                                  (#.+pcs-instance-tag+ . skip-pcs-instance)
```

```
                                    (#.+pcs-slot-unbound-tag+ . skip-pcs-slot-unbound)
                                    (#.+pcs-instance-stub-tag+ . skip-pcs-instance-stub))
  do (setf (aref table code) (symbol-function symbol)))
    (setf (aref table +true-tag+) #'(lambda (x) (declare (ignore x)) t))
    (setf (aref table +false-tag+) #'(lambda (x) (declare (ignore x)) nil))
    (setf *op-code-skip-table* table)))

(initialize-op-code-table)

(initialize-op-code-skip-table)

(defun obtain-version-number ()
  *version*)

(defun bin-dump (thing filename &key (dump-function 'dump-thing))
  (with-open-file (str filename :direction :output :element-type '(unsigned-byte 8)
       :if-exists :supersede)
    (write-byte (obtain-version-number) str)
    (let ((*print-readably* t))  ; so dump-other will signal errors
      (funcall dump-function thing str))
    (write-byte +end-tag+ str)))

(defmacro with-output-to-bin-dump ((stream-var filename) &body body)
  `(with-open-file (,stream-var ,filename :direction :output :element-type '(unsigned-byte\
 8)
                :if-exists :supersede)
     (write-byte (obtain-version-number) ,stream-var)
     (let ((*print-readably* t))  ; so dump-other will signal errors
       ,@body)
     (write-byte +end-tag+ ,stream-var)))

(defun read-bin-dump (filename)
  (with-open-file (str filename :direction :input :element-type '(unsigned-byte 8))
    (let ((version (read-byte str))
  (object (read-thing str))
  (end-tag? (read-byte str)))
      (if (eq end-tag? +end-tag+)
  (values object version)
  (error "File corrupted or saved improperly; end-tag not found."))))))

(defmacro with-input-from-bin-dump ((stream-var filename &key (full-read-p nil)) &body bod\
y)
  `(with-open-file (,stream-var ,filename :direction :input :element-type '(unsigned-byte \
8))
     (let ((version (read-byte ,stream-var)))
       version  ; ignorable
       ,@body
       ,(when full-read-p
          `(if (eq (read-byte ,stream-var) +end-tag+)
     version
     (error "File corrupted or saved improperly; end-tag not found."))))))
```

11.3.2 code/pcs/class-info.lisp

```
;;; -*- Mode: lisp; Syntax: ANSI-Common-Lisp; Base: 10; Package: PCS -*-
;;;
;;; Copyright 1998-99 Christopher R. Vincent (cvince@mit.edu)
;;; All Rights Reserved.
;;;
;;;------------------------------------------------------------------
;;;
;;; PERSISTENT CLASS INFORMATION
;;;

(in-package :pcs)
```

```
(defvar *class-info-table* (make-hash-table)
  "Maps class name to persisent class information.")

(defgeneric register-persistent-class (class &key if-exists)
  (:documentation "Register a persistent class. IF-EXISTS may be :error or :replace."))

(defmethod register-persistent-class ((class-name symbol) &key (if-exists :replace))
  (when (and (gethash class-name *class-info-table*)
             (eq if-exists :error))
    (error "Class ~S already registered as a persistent class." class-name))
  (setf (gethash class-name *class-info-table*)
        (make-instance 'class-info :name class-name)))

(defmethod register-persistent-class ((class class) &key (if-exists :replace))
  (register-persistent-class (class-name class) :if-exists if-exists))

(defgeneric register-persistent-slot (class slot-name &key readers writers force-persisten\
t)
  (:documentation "Register a slot for a persistent class. Merge the slot specification wi\
th
                   those inherited from direct superiors.  If FORCE-PERSISTENT is NIL
                   and persistence is not inherited, nothing is registered."))

(defmethod register-persistent-slot ((class class) (slot-name symbol)
                                     &key readers writers force-persistent)
  (let* ((class-name (class-name class))
         (class-info (cond ((gethash class-name *class-info-table*))
                           (t (error "No persistent class information registered for class\
 ~S"class-name))))
         (persistent-p force-persistent))
    ; Inherit slot attributes from superiors
    (loop for superclass in (class-direct-superclasses class)
          for parent-class-info = (get-class-info superclass)
          for parent-slot-info = (when parent-class-info
                                   (get-slot-info parent-class-info slot-name))
          when parent-slot-info
          do (setq readers (nunion readers (slot-info-readers parent-slot-info)))
          (setq writers (nunion writers (slot-info-writers parent-slot-info)))
          ; Note that we have inherited slot persitence.
          (unless persistent-p (setq persistent-p t)))
    (when persistent-p
      (setf (getf (class-info-slots class-info) slot-name)
            (make-instance 'slot-info
              :name slot-name
              :readers readers
              :writers writers)))))

(defun slot-idx (class-info slot-name)
  "Used to access persistent-instance-modification-state vectors.
   Must be called after all persistent slots have been registered for class-info."
  (check-type class-info class-info)
  (check-type slot-name symbol)
  (/ (position slot-name (class-info-slots class-info)) 2))

(defgeneric get-class-info (class &key error-p)
  (:documentation "Return the info for a persistent class."))

(defmethod get-class-info ((class-name symbol) &key (error-p nil))
  (let ((info (gethash class-name *class-info-table*)))
    (cond (info)
          (error-p (error "Class ~S not registered as a persistent class." class-name))
          (t nil))))

(defmethod get-class-info ((class class) &key (error-p nil))
  (get-class-info (class-name class) :error-p error-p))

(defgeneric get-slot-info (class-info slot &key error-p)
  (:documentation "Return the info for a persistent slot of a persistent class."))
```

```
(defmethod get-slot-info ((class-info class-info) (slot symbol) &key error-p)
  (let ((info (getf (class-info-slots class-info) slot)))
    (cond (info)
          (error-p (error "Slot ~S not registered in ~S." slot class-info))
          (t nil))))

(defun set-slot-info (class-info slot-name slot-info)
  (setf (getf (class-info-slots class-info) slot-name) slot-info))

(defsetf get-slot-info set-slot-info)

(defgeneric map-slot-info (function class-info)
  (:documentation "Enumerate the persistent slots for a class in slot-idx order."))

(defmethod map-slot-info ((function function) (class-info class-info))
  (loop for slot in (rest (class-info-slots class-info)) by #'cddr
        do (funcall function slot)))
```

11.3.3 code/pcs/class.lisp

```
;;; -*- Mode: lisp; Syntax: ANSI-Common-Lisp; Base: 10; Package: PCS -*-
;;;
;;; Copyright 1998-99 Christopher R. Vincent (cvince@mit.edu)
;;; All Rights Reserved.
;;;
;;;---------------------------------------------------------------------
;;;
;;; PERSISTENT CLASS STORAGE
;;;

(in-package :pcs)

(defclass identifier ()
    ((serial-number
       :type integer
       :initarg :serial-number
       :reader identifier-serial-number)
     (class-name
       :type symbol
       :initarg :class-name
       :reader identifier-class-name)
     (storage-mechanism-name
       :type symbol
       :initarg :storage-mechanism-name
       :reader identifier-storage-mechanism-name))
  (:documentation "A unique identifier for referring to a persistent object."))

(defclass storage-mechanism ()
    ((name
       :type symbol
       :initarg :name
       :accessor storage-mechanism-name))
    (:documentation "A storage mechanism, such as a database or distributed namespace."))

(defclass persistent-instance-mixin ()
    ((identifier
       :initarg :identifier
       :accessor persistent-instance-identifier)
     (cached-p
       :initarg :cached-p
       :initform nil
       :accessor persistent-instance-cached-p)
     (modified-p
       :initarg :modified-p
       :initform nil
       :accessor persistent-instance-modified-p)
```

```
     (slot-modification-state
       :initarg :slot-modification-state
       :accessor persistent-instance-slot-modification-state))
   (:documentation "Persistent classes must inherit from this mixin."))

(defmethod print-object ((instance persistent-instance-mixin) stream)
  (print-unreadable-object (instance stream :type t :identity t)
    (when (slot-boundp instance 'identifier)
      (with-slots (identifier) instance
(cond ((new-instance-identifier-p identifier)
       (write-string "NEW-INSTANCE" stream))
      (t (format stream "SN:~S" (identifier-serial-number (persistent-instance-identifier\
 instance)))))))))

(defclass class-info ()
  ((name
    :initarg :name
    :accessor class-info-name)
   (slots
    :initform nil
    :initarg :slots
    :accessor class-info-slots
    :documentation "A p-list mapping slot-names to slot-info instances.")
   (slot-count
    :initform nil
    :initarg :slot-count
    :accessor class-info-slot-count
    :documentation "The number of slots."))
  (:documentation "Information concerning a persistent class."))

(defclass slot-info ()
  ((name
    :initarg :name
    :accessor slot-info-name)
   (readers
    :initarg :readers
    :accessor slot-info-readers)
   (writers
    :initarg :writers
    :accessor slot-info-writers))
  (:documentation "Information concerning a slot for a persistent class."))
```

11.3.4 code/pcs/filesystem-storage.lisp

```
;;; -*- Mode: lisp; Syntax: ANSI-Common-Lisp; Base: 10; Package: PCS -*-
;;;
;;; Copyright 1998-99 Christopher R. Vincent (cvince@mit.edu)
;;; All Rights Reserved.
;;;
;;;------------------------------------------------------------------
;;;
;;; PORTABLE FILESYSTEM STORAGE MECHANISM
;;;

;; This isn't too robust with regard to persistence in the face of
;; system failures.  The :supersede keyword should cover individual saves,
;; but atomic multiple saves need transactions.
;; Log based would be faster for writes.

(in-package :pcs)

(defclass filesystem-storage-mechanism (storage-mechanism)
    ((pathname
       :initarg :pathname
       :reader storage-mechanism-pathname)
     (root-pathname
```

```
      :initarg :root-pathname
      :reader storage-mechanism-root-pathname))
  (:documentation "A storage mechanism that uses the filesystem to store objects."))

(defmethod allocate-storage-mechanism ((name symbol) (storage-mechanism (eql 'filesystem-s\
torage-mechanism))
                                       &key pathname)
  (make-instance storage-mechanism
    :pathname pathname
    :root-pathname (merge-pathnames "root" pathname)))

(defmethod save-instance-using-storage-mechanism ((instance persistent-instance-mixin)
(storage-mechanism filesystem-storage-mechanism)
&key if-exists)
  (declare (ignore if-exists))
  (let* ((identifier (persistent-instance-identifier instance))
         (serial-number-string (write-to-string (identifier-serial-number identifier)))
         (pathname (merge-pathnames serial-number-string (storage-mechanism-pathname stora\
ge-mechanism))))
    (bin-dump instance pathname :dump-function 'dump-pcs-instance)))

(defmethod save-instances-using-storage-mechanism ((instances list)
 (storage-mechanism filesystem-storage-mechanism)
 &key if-exists)
  (let ((local-instances
  (loop for instance in instances
for identifier = (persistent-instance-identifier instance)
while (eq (identifier-storage-mechanism identifier) storage-mechanism)
                collect instance)))
    (loop for instance in local-instances
          do (save-instance-using-storage-mechanism instance storage-mechanism :if-exists \
if-exists))
    (cond ((< (length local-instances) (length instances))
           (save-instances-using-storage-mechanism (subseq instances (length local-instanc\
es) (length instances))
                                                   storage-mechanism :if-exists if-exists))
          (t local-instances))))

(defmethod load-instance-using-storage-mechanism ((identifier identifier)
  (storage-mechanism filesystem-storage-mechanism))
  ; The load-form takes care of allocating an instance through the cache and filling its s\
lots.
  ; If an instance has already been allocated, this should change its slot values.
  (let* ((serial-number-string (write-to-string (identifier-serial-number identifier)))
         (pathname (merge-pathnames serial-number-string (storage-mechanism-pathname stora\
ge-mechanism))))
    (read-bin-dump pathname)))

(defmethod save-root-using-storage-mechanism (thing (storage-mechanism filesystem-storage-\
mechanism))
  (typecase thing
    (persistent-instance-mixin
     (bin-dump thing (storage-mechanism-root-pathname storage-mechanism)
               :dump-function 'dump-pcs-instance))
    (t (bin-dump thing (storage-mechanism-root-pathname storage-mechanism)))))

(defmethod load-root-using-storage-mechanism ((storage-mechanism filesystem-storage-mechan\
ism))
  (let ((pathname (storage-mechanism-root-pathname storage-mechanism)))
    (cond ((probe-file pathname)
           (read-bin-dump pathname))
          (t nil))))
```

11.3.5 code/pcs/identifier.lisp

```
;;; -*- Mode: lisp; Syntax: ANSI-Common-Lisp; Base: 10; Package: PCS -*-
```

```
;;;
;;; Copyright 1998-99 Christopher R. Vincent (cvince@mit.edu)
;;; All Rights Reserved.
;;;
;;;---------------------------------------------------------------------
;;;
;;; IDENTIFIERS
;;;

(in-package :pcs)

(defmethod print-object ((instance identifier) stream)
  (print-unreadable-object (instance stream :type t :identity t)
    (when (and (slot-boundp instance 'serial-number)
       (slot-boundp instance 'class-name)
       (slot-boundp instance 'storage-mechanism-name))
      (with-slots (serial-number class-name storage-mechanism-name) instance
(format stream "SN:~S CLASS:~S ~S" serial-number class-name storage-mechanism-name)))))

(defmethod identifier-storage-mechanism ((identifier identifier))
  (find-storage-mechanism (slot-value identifier 'storage-mechanism-name)))

(defgeneric allocate-identifier (serial-number class-name storage-mechanism)
  (:documentation "Allocate an identifier with the given attributes."))

(defmethod allocate-identifier ((serial-number integer) (class-name symbol) (storage-mecha\
nism symbol))
  (make-instance 'identifier
 :serial-number serial-number
 :class-name class-name
 :storage-mechanism-name storage-mechanism))

(defmethod allocate-identifier ((serial-number integer) (class-name symbol) (storage-mecha\
nism storage-mechanism))
  (allocate-identifier serial-number class-name (storage-mechanism-name storage-mechanism)\
))

(defgeneric allocate-new-identifier (class storage-mechanism)
  (:documentation "Allocate a new unique identifier with the given object class and storag\
e-mechanism."))

(defmethod allocate-new-identifier ((class symbol) (storage-mechanism symbol))
  (allocate-identifier (allocate-new-serial-number) class storage-mechanism))

(defmethod allocate-new-identifier (class (storage-mechanism storage-mechanism))
  (allocate-new-identifier (class-name class) (storage-mechanism-name storage-mechanism)))

(defclass new-instance-identifier (identifier) ()
  (:documentation "An identifier for an object that has yet to be saved into persistent st\
orage."))

(defmethod print-object ((instance new-instance-identifier) stream)
  (print-unreadable-object (instance stream :type t :identity t)))

(defparameter *new-instance-identifier* (make-instance 'new-instance-identifier)
  "This identifier is used as a dummy identifier until an object is saved the first time.")

(defgeneric new-instance-identifier-p (identifier)
  (:documentation "Determine if an identifier is a new-instance-identifier."))

(defmethod new-instance-identifier-p ((identifier identifier))
  nil)

(defmethod new-instance-identifier-p ((identifier new-instance-identifier))
  t)

(defmethod make-load-form ((instance identifier))
  "Initialization form guaranteed to be nil."
  (with-slots (serial-number class-name storage-mechanism-name) instance
```

```
    `(allocate-identifier ,serial-number ',class-name ',storage-mechanism-name)))

(defmethod make-load-form ((identifier new-instance-identifier))
  (error "Cannot make a load-form for identifier ~S, it represents a persistent object tha\
t has not been saved." identifier))

(defparameter *id-to-string-delim* #\/)

(defun identifier-to-string (identifier)
  "Compute a parsable string representation of an identifier.  Safe for HTTP URI encoding."
  (check-type identifier identifier)
  (with-slots (serial-number class-name storage-mechanism-name) identifier
    (format nil "~A~A~A~A~A~A~A~D"
            (short-package-name (symbol-package storage-mechanism-name))
            *id-to-string-delim*
            (symbol-name storage-mechanism-name)
            *id-to-string-delim*
            (short-package-name (symbol-package class-name))
            *id-to-string-delim*
            (symbol-name class-name)
            *id-to-string-delim*
            serial-number)))

;; This isn't very fault-tolerant.
(defun string-to-identifier (string)
  "Parse an identifier from a string created with identifier-to-string."
  (destructuring-bind (sm-package sm-name class-package class-name serial-number)
                      (loop for c across string
                            for idx upfrom 0
                            with prev-idx = 0
                            with len = (length string)
                            with max-idx = (1- len)
                            when (eq c *id-to-string-delim*)
                            collect (subseq string prev-idx idx)
                            and do (setq prev-idx (1+ idx))
                            else when (= idx max-idx)
                            collect (subseq string prev-idx len))
    (allocate-identifier (parse-integer serial-number)
                         (intern class-name class-package)
                         (intern sm-name sm-package))))
```

11.3.6 code/pcs/java-serialization.lisp

```
;;; -*- Mode: lisp; Syntax: ANSI-Common-Lisp; Base: 10; Package: PCS -*-
;;;
;;; Copyright 1998-99 Christopher R. Vincent (cvince@mit.edu)
;;; All Rights Reserved.
;;;
;;;------------------------------------------------------------------
;;;
;;; SUPPORT FOR JAVA SERIALIZATION
;;;

(in-package :pcs)

(defvar *max-java-serialization-depth* 0)

(defmethod instance-for-serialization ((instance persistent-instance-mixin) depth)
  "Only top-level objects are written, slots are filled with their identifier."
  ; Precache slots so that serialization can use slot-value.
  (unless (persistent-instance-cached-p instance)
    (cache-instance-slots instance))
  (cond ((<= depth *max-java-serialization-depth*) instance)
        (t (persistent-instance-identifier instance))))
```

11.3.7 code/pcs/log-based-storage.lisp

```
;;; -*- Mode: lisp; Syntax: ANSI-Common-Lisp; Base: 10; Package: PCS -*-
;;;
;;; Copyright 1998-99 Christopher R. Vincent (cvince@mit.edu)
;;; All Rights Reserved.
;;;
;;;---------------------------------------------------------------------
;;;
;;; PORTABLE LOG-BASED STORAGE MECHANISM
;;;

(in-package :pcs)

(defparameter *checkpoint-counter-reset* 524288)

(defclass log-based-storage-mechanism (storage-mechanism)
    ((pathname
      :initarg :pathname
      :reader storage-mechanism-pathname)
     (temp-pathname
      :initarg :temp-pathname
      :reader storage-mechanism-temp-pathname
      :documentation "Temporary pathname used when compacting instance log.")
     (root-pathname
      :initarg :root-pathname
      :reader storage-mechanism-root-pathname)
     (checkpoint-pathname
      :initarg :checkpoint-pathname
      :reader storage-mechanism-checkpoint-pathname)
     (checkpoint-counter
      :initarg :checkpoint-counter
      :initform 0
      :accessor storage-mechanism-checkpoint-counter
      :documentation "Dump a new checkpoint after *checkpoint-counter-reset* bytes.")
     (initialized-p
      :initarg :initialized-p
      :initform nil
      :accessor storage-mechanism-initialized-p)
     (offset-table
      :initarg :offset-table
      :reader storage-mechanism-offset-table)
     (file-stream
      :initarg :file-stream
      :initform nil)
     (lock
      :initarg :lock
      :initform (make-lock "Log-Based Object Storage.")
      :accessor storage-mechanism-lock))
  (:documentation "A storage mechanism that uses a file log to store objects."))

(defmethod allocate-storage-mechanism ((name symbol) (storage-mechanism (eql 'log-based-st\
orage-mechanism))
                                       &key pathname)
  (make-instance storage-mechanism
    :pathname pathname
    :temp-pathname (merge-pathnames (concatenate 'string (pathname-name pathname) "-copy")\
 pathname)
    :root-pathname (merge-pathnames "log-root" pathname)
    :checkpoint-pathname (merge-pathnames "checkpoint" pathname)
    :offset-table (make-hash-table)))

(defmethod storage-mechanism-file-stream ((storage-mechanism log-based-storage-mechanism))
  (with-slots (file-stream) storage-mechanism
    (cond ((and file-stream (open-stream-p file-stream)) file-stream)
          (t (setf file-stream (open (storage-mechanism-pathname storage-mechanism)
                                     :direction :io
                                     :element-type '(unsigned-byte 8)
                                     :if-exists :overwrite
                                     :if-does-not-exist :create))))))
```

```
(defun generate-log-based-storage-checkpoint (storage-mechanism)
  "Write a checkpoint file for faster initialization."
  (let ((stream (storage-mechanism-file-stream storage-mechanism)))
    (with-slots (offset-table checkpoint-pathname) storage-mechanism
    (bin-dump (list offset-table (file-length stream)) checkpoint-pathname))))

(defun load-log-based-storage-checkpoint (storage-mechanism)
  "Load checkpoint information, return number of log bytes the checkpoint captured."
  (with-slots (offset-table checkpoint-pathname) storage-mechanism
    (cond ((probe-file checkpoint-pathname)
           (destructuring-bind (table length) (read-bin-dump checkpoint-pathname)
             (setf offset-table table)
             length))
          (t (clrhash offset-table)
             0))))

(defun %initialize-log-based-storage (storage-mechanism)
  "Scan the log, noting the newest versions of persistent instances."
  (check-type storage-mechanism log-based-storage-mechanism)
  (let* ((stream (storage-mechanism-file-stream storage-mechanism))
         (max-position (1- (file-length stream)))
         (start (load-log-based-storage-checkpoint storage-mechanism))
         (offset-table (storage-mechanism-offset-table storage-mechanism)))
    (file-position stream start)
    (loop with position = start
          while (< position max-position)
          for serial-number = (skip-thing stream)
          for end = (file-position stream)
          do (check-type serial-number integer)
          do (setf (gethash serial-number offset-table)
                   (cons position (- end position)))
          (setq position end)))
  (setf (storage-mechanism-initialized-p storage-mechanism) t))

(defun %compact-log-based-storage (storage-mechanism &key (verbose-p t))
  "Compact the instance log, expunging obsolete versions."
  (check-type storage-mechanism log-based-storage-mechanism)
  (unless (storage-mechanism-initialized-p storage-mechanism)
    (%initialize-log-based-storage storage-mechanism))
  (when verbose-p
    (format t "~%; Compacting instance log..."))
  (with-slots (pathname temp-pathname offset-table) storage-mechanism
    (let* ((stream (storage-mechanism-file-stream storage-mechanism))
           (old-length (file-length stream))
           new-length)
      (file-position stream 0)
      (with-open-file (temp-stream temp-pathname
                                   :direction :output
                                   :element-type '(unsigned-byte 8)
                                   :if-exists :supersede
                                   :if-does-not-exist :create)
        (loop for serial-number being the hash-key of offset-table
              for (start . length) being the hash-value of offset-table
              with temp-position = 0
              with new-offset-table = (make-hash-table)
              do (file-position stream start)
              (loop repeat length do (write-byte (read-byte stream) temp-stream))
              (setf (gethash serial-number new-offset-table)
                    (cons temp-position length))
              (setq temp-position (file-position temp-stream))
              finally (setf offset-table new-offset-table
                            new-length (file-length temp-stream))))
      (close stream)
      (rename-file temp-pathname pathname :if-exists :supersede)
      (generate-log-based-storage-checkpoint storage-mechanism)
      (when verbose-p
        (let ((delta (- old-length new-length)))
          (format t "~%; Expunged ~,1F K (~,1F%)" (/ delta 1024) (* 100 (/ delta old-lengt\
```

```
h)))))))

(defmethod optimize-storage-mechanism-using-storage-mechanism ((storage-mechanism log-base\
d-storage-mechanism)
                                                               &key verbose)
  (with-lock-held ((storage-mechanism-lock storage-mechanism) :write)
    (%compact-log-based-storage storage-mechanism :verbose-p verbose)))

(defun %save-instance (instance storage-mechanism &key if-exists)
  (declare (ignore if-exists))
  (let* ((stream (storage-mechanism-file-stream storage-mechanism))
         (position (file-length stream))
         (offset-table (storage-mechanism-offset-table storage-mechanism))
         (serial-number (identifier-serial-number (persistent-instance-identifier instance\
))))
    (file-position stream position)
    (dump-pcs-instance instance stream)
    (let ((length (- (file-position stream) position)))
      (setf (gethash serial-number offset-table) (cons position length))
      (with-slots (checkpoint-counter) storage-mechanism
        (incf checkpoint-counter length)
        (when (> checkpoint-counter *checkpoint-counter-reset*)
          (generate-log-based-storage-checkpoint storage-mechanism)
          (setf checkpoint-counter 0))))
    instance))

(defmethod save-instance-using-storage-mechanism ((instance persistent-instance-mixin)
  (storage-mechanism log-based-storage-mechanism)
  &key if-exists)
  (declare (ignore if-exists))
  (with-lock-held ((storage-mechanism-lock storage-mechanism) :write)
    (unless (storage-mechanism-initialized-p storage-mechanism)
      (%initialize-log-based-storage storage-mechanism))
    (%save-instance instance storage-mechanism)))

(defun %save-instances (instances storage-mechanism &key if-exists)
  (let ((local-instances
         (loop for instance in instances
               for identifier = (persistent-instance-identifier instance)
               while (eq (identifier-storage-mechanism identifier) storage-mechanism)
               collect instance)))
    (loop for instance in local-instances
          do (%save-instance instance storage-mechanism))
    (cond ((< (length local-instances) (length instances))
           (%save-instances (subseq instances (length local-instances) (length instances))\
 storage-mechanism
                            :if-exists if-exists))
          (t local-instances))))

(defmethod save-instances-using-storage-mechanism ((instances list)
   (storage-mechanism log-based-storage-mechanism)
   &key if-exists)
  (with-lock-held ((storage-mechanism-lock storage-mechanism) :write)
    (unless (storage-mechanism-initialized-p storage-mechanism)
      (%initialize-log-based-storage storage-mechanism))
    (%save-instances instances storage-mechanism :if-exists if-exists)))

(defmethod load-instance-using-storage-mechanism ((identifier identifier)
  (storage-mechanism log-based-storage-mechanism))
  ; Use write mode locking since the storage-mechanism object is modified.
  (with-lock-held ((storage-mechanism-lock storage-mechanism) :write)
    (unless (storage-mechanism-initialized-p storage-mechanism)
      (%initialize-log-based-storage storage-mechanism))
    (let ((stream (storage-mechanism-file-stream storage-mechanism))
          (offset-table (storage-mechanism-offset-table storage-mechanism))
          (serial-number (identifier-serial-number identifier)))
      (file-position stream (car (gethash serial-number offset-table)))
      (read-thing stream))))
```

```
(defmethod save-root-using-storage-mechanism (thing (storage-mechanism log-based-storage-m\
echanism))
  (with-lock-held ((storage-mechanism-lock storage-mechanism) :write)
    (typecase thing
      (persistent-instance-mixin
       (bin-dump thing (storage-mechanism-root-pathname storage-mechanism)
                 :dump-function 'dump-pcs-instance))
      (t (bin-dump thing (storage-mechanism-root-pathname storage-mechanism))))))

(defmethod load-root-using-storage-mechanism ((storage-mechanism log-based-storage-mechani\
sm))
  (with-lock-held ((storage-mechanism-lock storage-mechanism) :write)
    (let ((pathname (storage-mechanism-root-pathname storage-mechanism)))
      (cond ((probe-file pathname)
             (read-bin-dump pathname))
            (t nil)))))
```

11.3.8 code/pcs/make-load-form.lisp

```
;;; -*- Mode: lisp; Syntax: ANSI-Common-Lisp; Base: 10; Package: PCS -*-
;;;
;;; Copyright 1998-99 Christopher R. Vincent (cvince@mit.edu)
;;; All Rights Reserved.
;;;
;;;---------------------------------------------------------------------
;;;
;;; MAKE-LOAD-FORM METHOD OF SAVING OBJECTS, DEPRICATED
;;;

(in-package :pcs)

;; The make-load-form method on persistent classes is reserved for optional use by storage\
-mechanisms.
;; The initialization form is guaranteed to be nil.

(defun set-slot-value-if-unmodified (instance slot-name value)
  "A utility function used by load-forms to fill unmodified slots from persistent storage."
  (unless (slot-modified-p instance slot-name)
    (setf (slot-value instance slot-name) value)))

(defun compute-persistent-list-form (list)
  "Compute a form for bindumping a list of valid persistent slot values."
  `(list ,@(loop for item in list
         collect (compute-persistent-value-form item))))

(defun compute-persistent-hash-table-form (hash-table)
  "Compute a form for bindumping a hash-table of valid persistent slot values."
  (let ((setf-forms (loop for key being each hash-key in hash-table
                          and value being each hash-value in hash-table
                          collect `(gethash ,(compute-persistent-value-form key) new-hash-\
table)
                          collect (compute-persistent-value-form value))))
    `(let ((new-hash-table (make-hash-table
                            :rehash-size ,(hash-table-rehash-size hash-table)
                            :rehash-threshold ,(hash-table-rehash-threshold hash-table)
                            :size ,(hash-table-size hash-table)
                            :test #',(hash-table-test hash-table))))
       ,@(when setf-forms (list `(setf ,@setf-forms)))
       new-hash-table)))

;; This does not check for invalid types.
(defun compute-persistent-value-form (value)
  "Compute a form for bindumping a persistent-slot value."
  (typecase value
    (list (compute-persistent-list-form value))
```

```
    (hash-table (compute-persistent-hash-table-form value))
    (identifier (make-load-form value))
    (persistent-instance-mixin
     `(allocate-instance-for-identifier ,(make-load-form (persistent-instance-identifier v\
alue))))
    (t `',value)))

;; Make sure this preserves unbound slots.
;; This could reduce dump size further by calling class-specific functions.
(defun define-make-load-form-method (class-info)
  "Define a make-load-form method for a class that is appropriate for use with persistent \
storage."
  (flet ((process-slot-info (slot-info)
           (with-slots (name type) slot-info
             `(when (slot-boundp self ',name)
                (list `(set-slot-value-if-unmodified
                        self
                        ',',name
                        ,(compute-persistent-value-form (slot-value self ',name))))))))
    (declare (inline process-slot-info))
    (let ((name (class-info-name class-info))
          (update-forms))
      (map-slot-info #'(lambda (info) (push (process-slot-info info) update-forms)) class-\
info)
      (%defmethod 'make-load-form nil `((self ,name)) nil
                  (list ``(let ((self ,(compute-persistent-value-form self)))
                            ,@,@update-forms
                            self))))))
```

11.3.9 code/pcs/package.lisp

```
;;; -*- Mode: lisp; Syntax: ANSI-Common-Lisp; Base: 10; Package: cl-user; -*-
;;;
;;; Copyright 1998-99 Christopher R. Vincent (cvince@mit.edu)
;;; All Rights Reserved.
;;;
;;;---------------------------------------------------------------------
;;;
;;; PERSISTENT CLASS STORAGE
;;;

(defpackage persistent-class-storage
  (:nicknames pcs)
  (:use future-common-lisp cds-util jos)
  (:export
   "DEFINE-PERSISTENT-CLASS"
   "IDENTIFIER"
   "IDENTIFIER-TO-STRING"
   "LOAD-OBJECT"
   "LOAD-ROOT-OBJECT"
   "MAKE-PERSISTENT-INSTANCE"
   "*MAX-JAVA-SERIALIZATION-DEPTH*"
   "OBJECT-CACHED-P"
   "OBJECT-IDENTIFIER"
   "OBJECT-MODIFIED-P"
   "OBJECT-SAVED-P"
   "OPTIMIZE-STORAGE-MECHANISM"
   "SAVE-OBJECT"
   "SAVE-OBJECTS"
   "SAVE-ROOT-OBJECT"
   "STORAGE-MECHANISM"
   "STRING-TO-IDENTIFIER"))
```

11.3.10 code/pcs/serial-number.lisp

```
;;; -*- Syntax: ANSI-Common-Lisp; Base: 10; Package: PCS -*-
;;;
;;; Copyright 1998-99 Christopher R. Vincent (cvince@mit.edu)
;;; All Rights Reserved.
;;;
;;;---------------------------------------------------------------------
;;;
;;; ALLOCATING SERIAL NUMBERS
;;;

(in-package :pcs)

;;;---------------------------------------------------------------------
;;;
;;; PORTABLE FILESYSTEM IMPLEMENTATION
;;;

;; Is it thread-safe for two processes to read/write disjoint sets
;; of slots on the same instance?  jcma says yes, and that next-lock
;; isn't needed.

;; This isn't really an appropriate use of a task-queue.  Should spawn
;; a new process that periodically writes out any changes to the state.

(defclass serial-number-state (task-queue)
  ((next
    :initarg :next
    :initform 0
    :accessor next-serial-number)
   (next-lock
    :initarg :next-lock
    :initform (make-lock "Next Serial Number" :type :simple)
    :accessor next-serial-number-lock)
   (pathname
    :initarg pathname
    :accessor serial-number-state-pathname)
   (file-stream
    :initarg :file-stream
    :initform nil)
   (last-saved
    :initarg :last-saved
    :initform 0
    :accessor serial-number-state-last-saved
    :documentation "The last number saved to the filesystem."))
  (:documentation "Serial number generation state."))

(defmethod serial-number-state-file-stream ((state serial-number-state))
  (with-slots (file-stream) state
    (cond ((and file-stream (open-stream-p file-stream)) file-stream)
          (t (setf file-stream (open (serial-number-state-pathname state)
                                     :direction :io
                                     :element-type '(unsigned-byte 8)
                                     :if-exists :overwrite
                                     :if-does-not-exist :create))))))

(defvar *serial-number-state-pathname* nil
  "Location of file for storing the serial-number state.")

(defvar *serial-number-state* nil
  "State of the serial-number generator.")

(defparameter *serial-number-state-lock*
  (make-lock "Serial Number Generator" :type :multiple-reader-single-writer)
  "Lock for controlling access to allocate-new-serial-number.")

(defun initialize-serial-number-state (&optional (pathname *serial-number-state-pathname*))
  (setf *serial-number-state* (ensure-active-task-queue 'serial-number-state)
        (serial-number-state-pathname *serial-number-state*) pathname)
```

```
  (let ((stream (serial-number-state-file-stream *serial-number-state*)))
    (setf (next-serial-number *serial-number-state*)
          (loop for byte = (read-byte stream nil nil)
                for idx upfrom 0
                with value = 0
                while byte
                do (setq value (+ value (ash byte (* 8 idx))))
                finally (return value)))
    *serial-number-state*))

(defun save-serial-number-state (state)
  "Save serial-number state.  Executes from within a task queue."
  (let ((stream (serial-number-state-file-stream state))
        (next (with-lock-held ((next-serial-number-lock state) :read)
                (next-serial-number state))))
    (with-slots (last-saved) state
      (unless (= next last-saved)
        (file-position stream 0)
        (loop with value = next
              while (> value 0)
              do (write-byte (logand 255 value) stream)
              (setq value (ash value -8)))
        (setf last-saved next)))))

(defun allocate-new-serial-number ()
  "Allocate a new serial-number to identify a persistent object."
  (with-lock-held (*serial-number-state-lock* :write)
    (let ((state (or *serial-number-state* (initialize-serial-number-state))))
      (with-slots (next) state
        (let ((serial-number next))
          (with-lock-held ((next-serial-number-lock state) :write)
            (incf next))
          (push-task-queue state #'save-serial-number-state)
          serial-number)))))

(defun reset-serial-number ()
  "Reset the serial-number generator."
  (with-lock-held (*serial-number-state-lock* :write)
    (let ((state (or *serial-number-state* (initialize-serial-number-state))))
      (with-slots (next last-saved) state
        (clear-task-queue state)
        (setf next 0
              last-saved 0)
        (push-task-queue state #'save-serial-number-state)))))
```

11.3.11 code/pcs/storage-mechanism.lisp

```
;;; -*- Mode: lisp; Syntax: ANSI-Common-Lisp; Base: 10; Package: PCS -*-
;;;
;;; Copyright 1998-99 Christopher R. Vincent (cvince@mit.edu)
;;; All Rights Reserved.
;;;
;;;---------------------------------------------------------------------
;;;
;;; STORAGE MECHANISMS
;;;

(in-package :pcs)

;; Rather than one root object, the database interface could support a mini-filesystem.
;; This would allow things like serial numbers to be stored in the same db.

(defvar *storage-mechanism-table* (make-hash-table)
  "A mapping from storage-mechanism name to storage-mechanism.")

(defgeneric find-storage-mechanism (name &key error-p)
```

```
  (:documentation "Find a storage-mechanism by name."))

(defmethod find-storage-mechanism ((name symbol) &key (error-p t))
  (let ((storage-mechanism (gethash name *storage-mechanism-table*)))
    (cond (storage-mechanism)
  (error-p (error "Could not find storage-mechanism named ~S" name)))))

(defgeneric allocate-storage-mechanism (name class-name &key)
  (:documentation "Allocate and name an instance of a subclass of storage-mechanism.
                   New primary methods may ignore name, and are only responsible for retur\
ning a new instance."))

(defmethod allocate-storage-mechanism :around ((name symbol) (class-name symbol) &key)
  "This method takes care of naming and registering the new storage-mechanism."
  (let ((class (find-class class-name nil)))
    (cond ((and class (find 'storage-mechanism (class-precedence-list class) :key #'class-\
name))
   (let ((new-storage-mechanism (call-next-method)))
     (setf (storage-mechanism-name new-storage-mechanism) name
   (gethash name *storage-mechanism-table*) new-storage-mechanism)))
  (t (error (format nil "Class ~S is not a valid subclass of storage-mechanism." class-na\
me))))))

(defgeneric initialize-persistent-storage-mechanism-for-class (storage-mechanism class)
  (:documentation "Perform persistent storage-mechanism initializations for a class when i\
t is defined or redefined."))

(defmethod initialize-persistent-storage-mechanism-for-class ((storage-mechanism storage-m\
echanism) class)
  (declare (ignore class))
  t)

(defgeneric save-instance-using-storage-mechanism (instance storage-mechanism &key if-exis\
ts)
  (:documentation "Save a persistent object according to its identifier, using the specifi\
ed storage-mechanism.
                   The :if-exists keyword defaults to :overwrite, but may be specified as \
:error.
                   New primary methods are always passed a working identifier, and are onl\
y responsible for writing out a representation
                   of instance to persistent storage.  If the identifier does not already \
exist for the
                   storage-mechanism, a new object should be saved.  Otherwise overwrite t\
he existing object."))

(defgeneric save-instances-using-storage-mechanism (instances storage-mechanism &key if-ex\
ists)
  (:documentation "Save a multiple persistent objects as an atomic action."))

(defgeneric load-instance-using-storage-mechanism (identifier storage-mechanism)
  (:documentation "Load an object from storage, caching values for unmodified slots on the\
 instance.
                   New primary methods are only responsible for setting the values of unmo\
dified slots
                   to their persistent state."))

(defgeneric save-root-using-storage-mechanism (thing storage-mechanism)
  (:documentation "Save the root data for a storage-mechanism."))

(defgeneric load-root-using-storage-mechanism (storage-mechanism)
  (:documentation "Load the root data for a storage-mechanism."))

(defgeneric optimize-storage-mechanism-using-storage-mechanism (storage-mechanism &key ver\
bose)
  (:documentation "Perform time and/or space optimizations on a storage-mechanism."))

(defmethod optimize-storage-mechanism-using-storage-mechanism (storage-mechanism &key verb\
ose)
  (declare (ignore storage-mechanism verbose))
```

```
  t)

(defparameter *default-storage-mechanism* nil
  "The default storage mechanism for storing persistent objects.")

(defparameter *supported-storage-mechanisms* nil
  "The list of all storage mechanisms currently in use.")
```

11.3.12 code/pcs/system.lisp

```
;;; -*- Mode: lisp; Syntax: ANSI-Common-Lisp; Base: 10; Package: PCS -*-
;;;
;;; Copyright 1998-99 Christopher R. Vincent (cvince@mit.edu)
;;; All Rights Reserved.
;;;
;;;---------------------------------------------------------------------
;;;
;;; PERSISTENT CLASS STORAGE
;;;

(in-package :pcs)

;;;---------------------------------------------------------------------
;;;
;;; FUNCTIONAL VERSIONS OF CL MACROS
;;;

(defun %defclass (class direct-superiors slot-defs declarations)
  "Distasteful, but a functional version of defclass is useful."
  (let ((fn-body `(lambda ()
                    (defclass ,class ,direct-superiors
                      ,slot-defs
                      ,@declarations))))
    (funcall (compile nil fn-body)))
  (find-class class))

(defun %defmethod (method qualifier lambda-list doc-string body)
  "Distasteful, but a functional version of defmethod is useful."
  (let ((fn-body `(lambda ()
                    (defmethod ,method ,@(when qualifier (list qualifier)) ,lambda-list
                               ,@(when doc-string (list doc-string)) ,@body))))
    (funcall (compile nil fn-body))))

(defun %defsetf (accessor writer)
  "Distasteful, but a functional version of defsetf is useful."
  (let ((fn-body `(lambda ()
                    (defsetf ,accessor ,writer))))
    (funcall (compile nil fn-body))))

;;;---------------------------------------------------------------------
;;;
;;; SUPPORT
;;;

(defparameter *supported-persistent-slot-types*
      '(integer string symbol keyword identifier list hash-table)
  "Primitive types supported in addition to other persistent classes.
   LIST is a nested list of persistent instances, identifiers, and other primitive types.
   HASH-TABLE is a hash-table of persistent instance, identifiers, and other primitive typ\
es.")

(defgeneric valid-persistent-type-p (type)
  (:documentation "Determine if a type can be used for a persistent slot."))

(defmethod valid-persistent-type-p ((type symbol))
  (or (member type *supported-persistent-slot-types*)
```

```
     (let ((class (find-class type nil)))
(and class (valid-persistent-class-p class)))))

(defgeneric valid-persistent-class-p (class)
  (:documentation "Determine if a class is a valid persistent class."))

(defmethod valid-persistent-class-p ((class class))
  (member (find-class 'persistent-instance-mixin) (class-precedence-list class)))

(defmethod valid-persistent-class-p ((class-name symbol))
  (let ((class (find-class class-name)))
    (and class (valid-persistent-class-p class))))

(defun initialize-persistent-instance (class-name instance new-identifier)
  "Initialize the slots inherited from persistent-instance-mixin."
  (check-type class-name symbol)
  (check-type instance persistent-instance-mixin)
  (let* ((class-info (get-class-info class-name :error-p t))
         (slot-count (class-info-slot-count class-info)))
    (with-slots (identifier slot-modification-state) instance
      (setf identifier new-identifier
            slot-modification-state (make-array slot-count :element-type 'bit :initial-ele\
ment 0)))))

(defun mark-instance-unmodified (instance)
  "Mark an instance and all of its slots as unmodified."
  (check-type instance persistent-instance-mixin)
  (let ((modification-state (persistent-instance-slot-modification-state instance)))
    (loop for idx from 0 upto (1- (length modification-state))
  do (setf (elt modification-state idx) 0)))
  (setf (persistent-instance-modified-p instance) nil))

(defun mark-all-slots-modified (instance)
  "Mark an instance as having all its slots modified."
  (check-type instance persistent-instance-mixin)
  (let ((modification-state (persistent-instance-slot-modification-state instance)))
    (loop for idx from 0 upto (1- (length modification-state))
  do (setf (elt modification-state idx) 1)))
  (setf (persistent-instance-modified-p instance) t))

(defun slot-cache-hit-p (instance slot-idx)
  "Return true if a valid slot value is already cached."
  (check-type instance persistent-instance-mixin)
  (check-type slot-idx integer)
  (or (persistent-instance-cached-p instance)
      (and (persistent-instance-modified-p instance)
           (eq 1 (elt (persistent-instance-slot-modification-state instance) slot-idx)))
      (new-instance-identifier-p (persistent-instance-identifier instance))))

(defgeneric slot-modified-p (instance slot-name)
  (:documentation "Determine if a persistent slot has been modified.  Methods defined auto\
matically."))

(defun ensure-persistent-identifier (instance storage-mechanism if-exists)
  (check-type instance persistent-instance-mixin)
  "If the instance is unsaved assign a new identifier, performing if-exists check."
  (let ((identifier (persistent-instance-identifier instance)))
    (cond ((new-instance-identifier-p identifier)
   (let* ((class (class-of instance))
                  (new-identifier (allocate-new-identifier class storage-mechanism)))
             (setf (persistent-instance-identifier instance) new-identifier)
             (register-instance instance new-identifier)))
  ((eq if-exists :error)
   (error "Object ~S has already been assigned a unique identifier, saving it would overw\
rite
                  its persistent state." instance))
          (t))))

(defun cache-instance-slots (instance)
```

```
  (check-type instance persistent-instance-mixin)
  (when (not (persistent-instance-cached-p instance))
    (let* ((identifier (persistent-instance-identifier instance))
           (storage-mechanism (identifier-storage-mechanism identifier)))
      (load-instance-using-storage-mechanism identifier storage-mechanism))
    (setf (persistent-instance-cached-p instance) t)))

;;;---------------------------------------------------------------------
;;;
;;; BIN-DUMPER SUPPORT
;;;

(defun define-bin-dumper-methods (class-info)
  "Install bin-dumper support for a class."
  (let ((name (class-info-name class-info))
        (slot-names))
    (map-slot-info #'(lambda (info) (push (slot-info-name info) slot-names)) class-info)
    (%defmethod 'dump-instance-slot-values nil `((instance ,name) (stream stream)) nil
                (list `(loop for slot-name in ',slot-names
                             doing (cond ((slot-boundp instance slot-name)
                                          (dump-thing (slot-value instance slot-name) stre\
am))
                                         (t (dump-pcs-slot-unbound stream))))))
    (%defmethod 'read-instance-slot-values nil `((instance ,name) (stream stream)) nil
                (list `(loop for slot-name in ',slot-names
                             for slot-value = (read-thing stream)
                             unless (slot-modified-p instance slot-name)
                             do (cond ((unbound-slot-p slot-value)
                                       (slot-makunbound instance slot-name))
                                      (t (setf (slot-value instance slot-name) slot-value)\
)))
                      'instance))
    (%defmethod 'skip-instance-slot-values nil `((class-name (eql ',name)) (stream stream)\
) nil
                (list 'stream  ; ignorable
                      `(loop repeat ,(length slot-names)
                             do (skip-thing stream))))))

;;;---------------------------------------------------------------------
;;;
;;; DEFINING AND INSTANTIATING PERSISTENT CLASSES
;;;

(defun define-persistent-class-accessors (class-name class-info)
  "Define reader, writer, and slot-modified-p methods for a persistent class."
  (check-type class-name symbol)
  (check-type class-info class-info)
  (flet ((process-slot-info (slot-info)
           (with-slots (name readers writers) slot-info
             (let ((slot-idx (slot-idx class-info name)))
               (loop for reader in readers
                     do (%defmethod reader nil `((instance ,class-name))
                                    "Load all persistent slots on a cache miss, return req\
uested value."
                                    `((unless (slot-cache-hit-p instance ,slot-idx)
                                        (cache-instance-slots instance))
                                      (slot-value instance ',name))))
               (loop for writer in writers
                     for set-name = (intern (string-upcase (concatenate 'string "set-" (sy\
mbol-name writer))))
                     do (%defmethod set-name nil `((instance ,class-name) value)
                                    "Set value, update slot modification state."
                                    `((setf (slot-value instance ',name) value
                                            (persistent-instance-modified-p instance) t
                                            (elt (persistent-instance-slot-modification-st\
ate instance) ,slot-idx) 1)
                                      value))                                          \

                     (%defsetf writer set-name))
```

```
              (%defmethod 'slot-modified-p nil `((instance ,class-name) (slot-name (eql '\
,name))) nil
                          `((and (persistent-instance-modified-p instance)
                                 (eq 1 (elt (persistent-instance-slot-modification-state \
instance) ,slot-idx)))))))))
    (declare (inline process-slot-info))
    (map-slot-info #'process-slot-info class-info)))

(defun %define-persistent-class (class-name direct-superiors slot-defs declarations)
  "Filter the standard defclass syntax, in preparation for persistent storage."
  (flet ((ensure-precedence-list ()
           (cond ((and (not (member 'persistent-instance-mixin direct-superiors))
                       (notany 'valid-persistent-class-p direct-superiors))
                  `(persistent-instance-mixin ,@direct-superiors))
                 (t direct-superiors)))
         (process-slot-spec (slot-spec)
           (let* ((slot-plist (rest slot-spec))
                  (allocation (getf slot-plist :allocation))
                  (reader (getf slot-plist :reader))
                  (writer (getf slot-plist :writer))
                  (accessor (getf slot-plist :accessor))
                  (readers `(,@(when reader (list reader)) ,@(when accessor (list accesso\
r))))
                  (writers `(,@(when writer (list writer)) ,@(when accessor (list accesso\
r)))))
             (when (eq :persistent allocation)
               (remf slot-plist :allocation))
             (values `(,(first slot-spec) ,@slot-plist)
                     (list (first slot-spec)
                           readers
                           writers
                           (eq :persistent allocation)))))
         (initialize-slot-info (class slot-info-list)
           (let ((class-info (get-class-info class)))
             ; Register slots explicitly referenced.
             (loop for slot-info in slot-info-list
                   do (destructuring-bind (slot-name readers writers forced-persistent-p)\
 slot-info
                        (register-persistent-slot class slot-name
                                                  :readers readers
                                                  :writers writers
                                                  :force-persistent forced-persistent-p)))
             ; Look for and register slots not referenced but inherited.
             (loop for superclass in (class-direct-superclasses class)
                   for superclass-info = (get-class-info superclass)
                   when superclass-info
                   do (map-slot-info #'(lambda (slot-info)
                                         (let ((slot-name (slot-info-name slot-info)))
                                           (unless (get-slot-info class-info slot-name)
                                             ; Reuse slot-info instance from superior.
                                             (setf (get-slot-info class-info slot-name) s\
lot-info))))
                                     superclass-info)))))
    (declare (inline ensure-precedence-list process-slot-spec initialize-slot-info))
    (let* ((new-class-info (register-persistent-class class-name))
           new-slot-info-list
           (new-class (%defclass class-name (ensure-precedence-list)
                                 (loop for slot-spec in slot-defs
                                       collect (multiple-value-bind (slot-def slot-info)
                                                                    (process-slot-spec slo\
t-spec)
                                                 (setf new-slot-info-list (push slot-info \
new-slot-info-list))
                                                 slot-def))
                                 declarations)))
      (initialize-slot-info new-class new-slot-info-list)
      (setf (class-info-slot-count new-class-info)
            (/ (length (class-info-slots new-class-info)) 2))
      (define-persistent-class-accessors class-name new-class-info)
```

```
      ;; (define-make-load-form-method new-class-info)
      (define-bin-dumper-methods new-class-info)
      new-class)))

(defmacro define-persistent-class (class-name direct-superiors slot-defs &rest declaration\
s)
  "Call this instead of defclass when defining a persistent class."
  `(%define-persistent-class ',class-name ',direct-superiors ',slot-defs ',declarations))

(defun make-persistent-instance (class-name &rest args)
  "Call this instead of make-instance when instantiating a persistent class."
  (let ((new-instance (apply #'make-instance class-name args)))
    (initialize-persistent-instance class-name new-instance *new-instance-identifier*)
    (setf (persistent-instance-cached-p new-instance) t)
    (mark-all-slots-modified new-instance)
    new-instance))

;;;-------------------------------------------------------------------
;;;
;;; INSTANCE CACHE
;;;

(defparameter *persistent-instance-cache* (make-weak-hash-table :test #'eq)
  "A mapping from serial-number to object for all instantiated persistent objects.")

(defun purge-instance-cache ()
  "Purge the instance cache, useful for debugging.
   For correct behavior, no persistent instances should be referenced when purge-instance-\
cache is called."
  (weak-clrhash *persistent-instance-cache*)
  t)

; This could resource instances instead of calling make-instance.
(defun allocate-empty-instance (identifier)
  "Allocate an instance to copy a persistent object into."
  (check-type identifier identifier)
  (let* ((class-name (identifier-class-name identifier))
         (instance (make-instance class-name)))
    (initialize-persistent-instance class-name instance identifier)
    instance))

(defun register-instance (instance identifier)
  "Register an instance in the cache with its identifier."
  (check-type instance persistent-instance-mixin)
  (check-type identifier identifier)
  (let ((serial-number (identifier-serial-number identifier)))
    (setf (weak-gethash serial-number *persistent-instance-cache*) instance)))

(defun allocate-instance-for-identifier (identifier)
  "Allocate an instance of a persistent object.  If another process has already allocated
   an instance of the object, this returns the same object.  Storage mechanisms must use t\
his
   interface for instantiating a persistent object."
  (check-type identifier identifier)
  (let ((serial-number (identifier-serial-number identifier)))
    (cond ((weak-gethash serial-number *persistent-instance-cache*))
  (t (register-instance (allocate-empty-instance identifier) identifier)))))

;;;-------------------------------------------------------------------
;;;
;;; SAVING AND LOADING PERSISTENT INSTANCES
;;;

(defgeneric object-saved-p (instance)
  (:documentation "Determine if an object has been assigned a persistent identity."))

(defmethod object-saved-p ((instance persistent-instance-mixin))
  (not (new-instance-identifier-p (persistent-instance-identifier instance))))
```

```
(defgeneric object-cached-p (instance)
  (:documentation "Determine if an object is cached in dynamic memory."))

(defmethod object-cached-p ((instance persistent-instance-mixin))
  (persistent-instance-cached-p instance))

(defgeneric object-modified-p (instance)
  (:documentation "Determine if an object has been modified."))

(defmethod object-modified-p ((instance persistent-instance-mixin))
  (persistent-instance-modified-p instance))

(defgeneric save-object (instance &key storage-mechanism if-exists)
  (:documentation "Store a persistent object, storage-mechanism (used for unsaved objects)\
 defaults
   to *default-storage-mechanism*. IF-EXISTS may be :overwrite or :error. All persistent i\
nstances
   pointed to by this object must already have been saved at least once."))

(defmethod save-object ((instance persistent-instance-mixin)
&key (storage-mechanism *default-storage-mechanism*) (if-exists :overwrite))
  (let ((identifier (persistent-instance-identifier instance)))
    (cond ((new-instance-identifier-p identifier)
           (ensure-persistent-identifier instance storage-mechanism if-exists)
   (save-instance-using-storage-mechanism instance storage-mechanism :if-exists if-exists\
))
  (t (let ((previous-storage-mechanism (identifier-storage-mechanism identifier)))
       (save-instance-using-storage-mechanism instance previous-storage-mechanism :if-exi\
sts if-exists)))))
  (mark-instance-unmodified instance)
  instance)

(defgeneric save-objects (instances &key storage-mechanism if-exists)
  (:documentation "Store a list of persistent objects atomically, storage-mechanism (used \
for unsaved objects)
                  defaults to *default-storage-mechanism*.
                  IF-EXISTS may be :overwrite or :error."))

;; Checking persistent class validity here may be excessive.
(defmethod save-objects ((instances list)
 &key (storage-mechanism *default-storage-mechanism*) (if-exists :overwrite))
  (loop for instance in instances
doing (unless (valid-persistent-class-p (class-of instance))
(error "The object ~S is not a valid persistent object." instance)))
  (map nil #'(lambda (instance) (ensure-persistent-identifier instance storage-mechanism i\
f-exists))
       instances)
  (save-instances-using-storage-mechanism instances storage-mechanism :if-exists if-exists)
  (map nil 'mark-instance-unmodified instances)
  instances)

(defgeneric load-object (identifier &key force-cached-p)
  (:documentation "Retrieve a copy of a persistent object given an identifier, optionally \
forcing
                  slot values to be immediately cached."))

(defmethod load-object ((identifier identifier) &key force-cached-p)
  (let ((instance (allocate-instance-for-identifier identifier)))
    (when (and force-cached-p (not (persistent-instance-cached-p instance)))
      (cache-instance-slots instance))
    instance))

(defmethod load-object ((identifier new-instance-identifier) &key force-cached-p)
  (declare (ignore force-cached-p))
  (error "Attempt to load unsaved persistent object."))

(defgeneric object-identifier (instance)
  (:documentation "Return the identifier for a persistent object."))
```

```
(defmethod object-identifier ((instance persistent-instance-mixin))
  (let ((identifier (persistent-instance-identifier instance)))
    (when (new-instance-identifier-p identifier)
      (error "Attempt to access identifier for an unsaved persistent object."))
    identifier))

(defun optimize-storage-mechanism (&key (storage-mechanism *default-storage-mechanism*) ve\
rbose)
  "Perform time and/or space optimizations on a storage mechanism."
  (optimize-storage-mechanism-using-storage-mechanism storage-mechanism :verbose verbose))

;;;-------------------------------------------------------------------
;;;
;;; ROOT OBJECTS
;;;

(defgeneric save-root-object (thing &key storage-mechanism)
  (:documentation "Save the root PCS object for a storage-mechanism. This may be any valid\
 persistent type."))

(defmethod save-root-object (thing &key (storage-mechanism *default-storage-mechanism*))
  (save-root-using-storage-mechanism thing storage-mechanism)
  thing)

(defgeneric load-root-object (&key storage-mechanism)
  (:documentation "Load the root PCS object for a storage-mechanism, initially NIL."))

(defmethod load-root-object (&key (storage-mechanism *default-storage-mechanism*))
  (load-root-using-storage-mechanism storage-mechanism))
```

11.3.13 code/pcs/test.lisp

```
;;; -*- Mode: lisp; Syntax: ANSI-Common-Lisp; Base: 10; Package: PCS -*-
;;;
;;; Copyright 1998-99 Christopher R. Vincent (cvince@mit.edu)
;;; All Rights Reserved.
;;;
;;;-------------------------------------------------------------------
;;;
;;; PCS TEST
;;;

(in-package :pcs)

(defmacro do-pcs-test ((string) &rest body)
  `(progn
     (format t "~%; Testing ~A." ,string)
     (unless (progn ,@body)
       (error "PCS test failed: ~A" ,string))
     t))

(define-persistent-class foo ()
    ((a :initarg :a :type integer :accessor foo-a :allocation :persistent)
     (b :initarg :b :type string :accessor foo-b :allocation :persistent)
     (c :initarg :c :type list :accessor foo-c :allocation :persistent)
     (temp :initarg :temp :accessor foo-temp))
    (:documentation "A sample persistent class."))

;; Careful, this test function both bashes the PCS instance cache and pollutes the databas\
e.
;; This should test the correct dumping different slot types.
(defun test-pcs (&aux test-foo1 test-id1 test-foo2)
  (format t "~%; Test PCS start.")
  (do-pcs-test ("serial number generation")
               ; Quick test, wastes some serial numbers.
               (loop repeat 10
```

```
                    for sn = (allocate-new-serial-number)
                    unless (and (integerp sn)
                                (or (not record)
                                    (not (member sn record))))
                    do (return nil)
                    collect sn into record
                    finally (return t)))
  (do-pcs-test ("make instance")
               (and (setq test-foo1 (make-persistent-instance 'foo :a 45 :b "foo-string" :\
temp 'temp-value))
                    (new-instance-identifier-p (persistent-instance-identifier test-foo1))
                    (persistent-instance-cached-p test-foo1)
                    (persistent-instance-modified-p test-foo1)
                    (slot-modified-p test-foo1 'a)
                    (eq 45 (foo-a test-foo1))
                    (eq 'temp-value (foo-temp test-foo1))))
  (do-pcs-test ("save object")
               (and (save-object test-foo1)
                    (setq test-id1 (persistent-instance-identifier test-foo1))
                    (integerp (identifier-serial-number test-id1))
                    (eq 'foo (identifier-class-name test-id1))
                    (eq *default-storage-mechanism* (identifier-storage-mechanism test-id1\
))
                    (persistent-instance-cached-p test-foo1)
                    (not (persistent-instance-modified-p test-foo1))
                    (not (slot-modified-p test-foo1 'a))))
  (do-pcs-test ("load cached object")
               (eq test-foo1 (load-object test-id1)))
  (do-pcs-test ("load purged object")
               (and (purge-instance-cache)
                    (setq test-foo2 (load-object test-id1))
                    (not (persistent-instance-cached-p test-foo2))
                    (not (persistent-instance-modified-p test-foo2))
                    (not (slot-modified-p test-foo2 'a))))
  (do-pcs-test ("uncached, unmodified slot read")
               (and (eq 45 (foo-a test-foo2))
                    (persistent-instance-cached-p test-foo2)
                    (not (persistent-instance-modified-p test-foo2))
                    (not (slot-modified-p test-foo2 'a))))
  (do-pcs-test ("uncached, unmodified slot write")
               (and (purge-instance-cache)
                    (setq test-foo2 (load-object test-id1))
                    (setf (foo-a test-foo2) 123)
                    (not (persistent-instance-cached-p test-foo2))
                    (persistent-instance-modified-p test-foo2)
                    (slot-modified-p test-foo2 'a)))
  (do-pcs-test ("uncached, modified read")
               (and (foo-a test-foo2)
                    (not (persistent-instance-cached-p test-foo2))
                    (string-equal "foo-string" (foo-b test-foo2))
                    (persistent-instance-cached-p test-foo2)
                    (persistent-instance-modified-p test-foo2)
                    (slot-modified-p test-foo2 'a)
                    (eq 123 (foo-a test-foo2))
                    (not (slot-modified-p test-foo2 'b))))
  (format t "~%; Test PCS done.")
  t)
```

11.3.14 code/platform/genera/pcs/package.lisp

```
;;; -*- Mode: lisp; Syntax: ANSI-Common-Lisp; Base: 10; Package: cl-user; -*-
;;;
;;; Copyright 1998-99 Christopher R. Vincent (cvince@mit.edu)
;;; All Rights Reserved.
;;;
;;;-------------------------------------------------------------------
```

```
;;;
;;; PERSISTENT CLASS STORAGE
;;;
;;; -*- Mode: lisp; Syntax: ANSI-Common-Lisp; Base: 10; Package: cl-user; -*-
;;;
;;; Copyright 1998 Christopher R. Vincent (cvince@mit.edu)
;;; All Rights Reserved.
;;;
;;;---------------------------------------------------------------------
;;;
;;; PERSISTENT CLASS STORAGE
;;;

(import '(clos:class-precedence-list) :pcs)
```

11.3.15 code/platform/genera/pcs/serial-number.lisp

```
;;; -*- Syntax: ANSI-Common-Lisp; Base: 10; Package: PCS -*-
;;;
;;; Copyright 1998-99 Christopher R. Vincent (cvince@mit.edu)
;;; All Rights Reserved.
;;;
;;;---------------------------------------------------------------------
;;;
;;; ALLOCATING SERIAL NUMBERS
;;;

(in-package :pcs)

;;;---------------------------------------------------------------------
;;;
;;; GENERA-STATICE IMPLEMENTATION
;;;

(defvar *serial-number-state* nil
  "Next serial-number.")

(defun serial-number-not-initialized-error ()
  (error "Serial number state has not been initialized."))

(defparameter *serial-number-state-database-pathname* nil
  "Location of Statice database for storing the serial-number state.")

(statice:define-entity-type serial-number-state ()
  ((value integer)))

(statice:define-schema serial-number-schema (serial-number-state))

(defun get-serial-number-state ()
  (let ((found-p nil)
(entity))
    (statice:for-each ((state serial-number-state))
      (when found-p
(error "Multiple serial-number-state entities found in database ~S." *serial-number-state\
-database-pathname*))
      (setq entity state)
      (setq found-p t))
    (or entity
(error "Could not find serial-number-state entity in database ~S." *serial-number-state-d\
atabase-pathname*))))

(defun initialize-serial-number-state-database (if-not-initialized)
  (unless (probe-file *serial-number-state-database-pathname*)
    (statice:make-database *serial-number-state-database-pathname*
                           'serial-number-schema
                           :if-exists :error))
```

```
  (statice:with-database (db *serial-number-state-database-pathname*)
    (statice:with-transaction ()
      (let ((state (get-serial-number-state)))
        (cond (state (setf (serial-number-state-value state) 0))
              (t (make-serial-number-state :value 0)))))))

(defun allocate-new-serial-number (&key (if-not-initialized :initialize))
   "Allocate a new serial-number to identify a persistent object.
   IF-NOT-INITIALIZED may be :ERROR or :INITIALIZE."
   (flet ((read-state-from-database ()
            (let ((old-state (setf *serial-number-state*
                                   (serial-number-state-value (get-serial-number-state)))))
              (incf *serial-number-state*)
              old-state))
          (create-new-state-database ()
            (initialize-serial-number-state-database)
            (setf *serial-number-state* 1)
            0))
     (declare (inline read-state-from-database create-new-state-database))
     (cond (*serial-number-state*
            (let ((old-state *serial-number-state*))
              (incf *serial-number-state*
                    (statice:with-database (db *serial-number-state-database-pathname*)
                      (statice:with-transaction ()
                        (setf (serial-number-state-value (get-serial-number-state)) *seria\
l-number-state*)))
                    old-state)))
           ((probe-file *serial-number-state-database-pathname*)
            (read-state-from-database))
           (t (ecase if-not-initialized
                (:initialize (create-new-state-database))
                (:error (serial-number-not-initialized-error)))))))

(defun reset-serial-number ()
  "Reset the serial-number generator."
  (initialize-serial-number-state-database)
  (setf *serial-number-state* 0))
```

11.3.16 code/platform/genera/pcs/statice-storage.lisp

```
;;; -*- Mode: lisp; Syntax: ANSI-Common-Lisp; Base: 10; Package: PCS -*-
;;;
;;; Copyright 1998-99 Christopher R. Vincent (cvince@mit.edu)
;;; All Rights Reserved.
;;;
;;;------------------------------------------------------------------
;;;
;;; STATICE STORAGE MECHANISM
;;;

(in-package :pcs)

; This needs to be updated to reflect the new PCS API.

(defclass statice-storage-mechanism (storage-mechanism)
    ((pathname
       :initarg :pathname
       :reader storage-mechanism-pathname)
     (schema-name
       :initarg :schema-name
       :initform (gensym "SCHEMA-")
       :reader storage-mechanism-schema-name))
  (:documentation "A storage mechanism that uses a Statice database to store objects."))

(defmethod allocate-storage-mechanism ((name symbol) (storage-mechanism (eql 'statice-stor\
```

```
age-mechanism))
       &key pathname schema-name)
  (make-instance storage-mechanism
 :pathname pathname
 :schema-name schema-name))

(statice:define-entity-type pcs-entity ()
  ((serial-number integer :inverse pcs-entity-with-serial-number :unique t)
   (value t :reader pcs-entity-value :writer pcs-entity-value)))

; Figure out what should happen here.  Try to check for schema consistency to determine da\
tabase validity?
(defmethod finalize-persistent-storage-mechanism ((storage-mechanism statice-storage-mecha\
nism))
  (let ((fn-body '(lambda ()
    (statice:define-schema ,(storage-mechanism-schema-name storage-mechanism)
   (pcs-entity)))))
    (funcall (compile nil fn-body)))
  (cond ((y-or-n-p "Recreate the database ~S from scratch, losing all data? " (storage-mec\
hanism-pathname storage-mechanism))
(statice:make-database (storage-mechanism-pathname storage-mechanism)
       (storage-mechanism-schema-name storage-mechanism)
       :if-exists :rebuild))
(t (handler-case
     (statice:make-database (storage-mechanism-pathname storage-mechanism)
    (storage-mechanism-schema-name storage-mechanism)
    :if-exists :error)
     (dbfs:file-already-exists)))))

(defun get-statice-entity-with-identifier (identifier)
  "Inside a database transaction, retrieve the entity for an identifier."
  (pcs-entity-with-serial-number (identifier-serial-number identifier)))

;; It's okay to call this while already in a transaction, right?
(defmethod save-object-using-storage-mechanism ((instance persistent-object)
(storage-mechanism statice-storage-mechanism)
&key if-exists)
  (declare (ignore if-exists))
  (let ((identifier (persistent-object-identifier instance)))
    (statice:with-database (db (storage-mechanism-pathname storage-mechanism))
      (statice:with-transaction ()
(let ((entity (get-statice-entity-with-identifier identifier)))
  (unless entity
    (setq entity (make-pcs-entity :serial-number (identifier-serial-number identifier))))
  (setf (pcs-entity-value entity) instance)
  instance)))))

(defmethod save-objects-using-storage-mechanism ((instances list)
 (storage-mechanism statice-storage-mechanism)
 &key if-exists)
  (let ((local-instances
  (loop for instance in instances
for identifier = (persistent-object-identifier instance)
when (eq (identifier-storage-mechanism identifier) storage-mechanism)
  collect instance)))
    (statice:with-database (db (storage-mechanism-pathname storage-mechanism))
      (statice:with-transaction ()
(loop for instance in local-instances
      doing (save-object-using-storage-mechanism instance storage-mechanism :if-exists if\
-exists))
(save-objects-using-storage-mechanism (set-difference instances local-instances) storage-\
mechanism :if-exists if-exists)))
    local-instances))

(defmethod load-object-using-storage-mechanism ((identifier identifier)
(storage-mechanism statice-storage-mechanism))
  (statice:with-database (db (storage-mechanism-pathname storage-mechanism))
    (statice:with-transaction ()
      (let ((entity (get-statice-entity-with-identifier identifier)))
```

```
(unless entity
  (error "Could not find statice entity for object with identifier ~S" identifier))
; The load-form takes care of allocating an instance through the cache and filling its sl\
ots.
; If an instance has already been allocated, this should change its slot values.
(pcs-entity-value entity)))))
```

11.3.17 code/platform/genera/pcs/weak-hash-table.lisp

```
;;; -*- Syntax: ANSI-Common-Lisp; Base: 10; Package: PCS -*-
;;;
;;; Copyright 1998-99 Christopher R. Vincent (cvince@mit.edu)
;;; All Rights Reserved.
;;;
;;;------------------------------------------------------------------
;;;
;;; WEAK HASH TABLES
;;;

(in-package :pcs)

;; Set function cells instead of inlining for equivalents

(declaim (inline (weak-gethash set-weak-hash-table weak-gethash weak-maphash weak-clrhash)\
))

(defun make-weak-hash-table (&key (test #'eql))
  (scl:make-hash-table :test test :gc-protect-values nil :locking :process))

(defun weak-gethash (key weak-hash-table)
  (gethash key weak-hash-table))

(defun set-weak-hash-table (key weak-hash-table value)
  (setf (gethash key weak-hash-table) value))

(defsetf weak-gethash set-weak-hash-table)

(defun weak-maphash (function weak-hash-table)
  (maphash function weak-hash-table))

(defun weak-clrhash (weak-hash-table)
  (clrhash weak-hash-table))
```

11.4 Archive Framework

11.4.1 code/cds/archive.lisp

```
;;; -*- Mode: lisp; Syntax: ANSI-Common-Lisp; Base: 10; Package: CDS -*-
;;;
;;; Copyright 1998-99 Christopher R. Vincent (cvince@mit.edu)
;;; All Rights Reserved.
;;;
;;;------------------------------------------------------------------
;;;
;;; ARCHIVE INFRASTRUCTURE
;;;

(in-package :cds)
```

```
;;;---------------------------------------------------------------------
;;;
;;; LOCKS
;;;

(defgeneric get-lock (object)
  (:documentation "Return a lock for a lockable object."))

(defmethod get-lock ((object lockable-object-mixin))
  (cond ((%get-lock object))
        (t (setf (%get-lock object)
                 (make-lock (format nil "~S Lock" (class-name (class-of object)))
                            :type :multiple-reader-single-writer)))))

(defmacro with-resource-locked ((resource &optional (mode :write)) &body body)
  "Lock a resource for reading or writing."
  `(with-lock-held ((get-lock ,resource) ,mode)
     ,@body))

(defmacro with-resources-locked ((resources &optional (mode :write)) &body body)
  "Lock a list of resources, preserving order, for reading or writing."
  (cond ((> (length resources) 1)
         `(with-resource-locked (,(car resources) ,mode)
            (with-resources-locked (,(rest resources) ,mode)
              ,@body)))
        (t `(with-resource-locked (,(car resources) ,mode)
              ,@body))))

;;;---------------------------------------------------------------------
;;;
;;; ARCHIVES
;;;

(defvar *archive-table* nil
  "The top-level data structure for the development server.
   Maps keyword names to archives.")

(defparameter *archive-table-lock* (make-lock "CDS Archive Table"
                                              :type :multiple-reader-single-writer)
  "Lock for accessing the archive table.")

(defun purge-archive-table ()
  "Reset the initialization state of the archive table for debugging."
  (with-lock-held (*archive-table-lock* :write)
      (setf *archive-table* nil))
  t)

(defun save-archive-table ()
  "Save archive table state."
  (with-lock-held (*archive-table-lock* :write)
    (save-root-object *archive-table*)))

;; This assumes that we start out with a fresh PCS database (root object NIL).
(defun initialize-archive-table ()
  "Initialize the archive table state."
  (with-lock-held (*archive-table-lock* :write)
    (setf *archive-table* (or (load-root-object) (make-hash-table)))))

(defun register-archive (archive &key (if-exists :overwrite))
  "Register a (already saved) archive in the archive table.
   IF-EXISTS may be :overwrite or :error."
  (check-type archive archive)
  (with-lock-held (*archive-table-lock* :write)
    (unless *archive-table* (initialize-archive-table))
    (let ((name (archive-name archive)))
      (when (and (eq if-exists :error)
                 (gethash name *archive-table*))
        (error "Archive ~S is already registered." name))
```

```
      (setf (gethash name *archive-table*) archive)
      (save-archive-table))))

(defun get-archive (name &key (error-p nil))
  "Look up a archive."
  (check-type name keyword)
  (unless *archive-table*
    (with-lock-held (*archive-table-lock* :write)
      (initialize-archive-table)))
  (with-lock-held (*archive-table-lock* :read)
    (let* ((archive (gethash name *archive-table*)))
      (cond (archive)
            (error-p (error "Archive ~S not found." name))
            (t nil)))))

(defun map-archives (function)
  "Map over all registered archives."
  (check-type function function)
  (unless *archive-table*
    (with-lock-held (*archive-table-lock* :write)
      (initialize-archive-table)))
  (with-lock-held (*archive-table-lock* :read)
    (maphash #'(lambda (key value)
                 (declare (ignore key))
                 (funcall function value))
             *archive-table*)))

(defgeneric make-index-for-archive (archive)
  (:documentation "Create an index appropriate for an archive."))

(defgeneric index-resource (resource archive &key &allow-other-keys)
  (:documentation "Register a resource in the index for an archive, resource is not modifi\
ed."))

(defgeneric find-indexed-resource (archive spec &key error-p &allow-other-keys)
  (:documentation "Find resources in an archive according to spec and other keys."))

(defmethod find-indexed-resource ((archive identifier) spec &rest args)
  (apply 'find-indexed-resource (load-object archive) spec args))

(defgeneric map-indexed-resources (function archive &rest qualifiers)
  (:documentation "Map over indexed resources, constrained by qualifiers."))

(defgeneric register-resource (resource archive &key save-resource-p save-archive-p)
  (:documentation "Register a resource as a member of an archive, resource must be saved
    at least once before or during this process. If already saved, resource is not modifie\
d."))

(defmethod register-resource ((resource archived-resource) (archive archive)
                              &key (save-resource-p t)
                              (save-archive-p t))
  (when save-resource-p
    (save-object resource))
  (index-resource resource archive)
  (when save-archive-p
    (save-object archive)))

;;;---------------------------------------------------------------------
;;;
;;; RESOURCES
;;;

(defmethod archive-collections ((archive identifier))
  (archive-collections (load-object archive)))

(defgeneric resource-creation-time-string (resource)
  (:documentation "Cache and return the printed representation of a resource creation time\
."))
```

```
(defmethod resource-creation-time-string ((resource resource))
  (or (%resource-creation-time-string resource)
      (setf (%resource-creation-time-string resource)
            (write-time-to-string (resource-creation-time resource)))))

(defgeneric register-new-version (resource version &key save-version-p save-resource-p)
  (:documentation "Register and save a new version of a versioned resource.
   If SAVE-RESOURCE-P version must be saved at least once before or during this operation.\
"))

(defmethod register-new-version ((resource versioned-resource) (version resource-version)
                                 &key (save-version-p t) (save-resource-p t))
  (setf (resource-versions resource) (nconc (resource-versions resource) (list version))
        (resource-version-number version) (incf (resource-version-count resource)))
  (when save-version-p
    (save-object version))
  (when save-resource-p
    (save-object resource))
  version)

(defgeneric map-resource-versions (function resource)
  (:documentation "Map over the versions of a resource."))

(defmethod map-resource-versions ((function function) (resource versioned-resource))
  (map nil #'(lambda (version)
               (funcall function version))
       (resource-versions resource))
  resource)

(defgeneric get-resource-version (resource index &key error-p)
  (:documentation "Get a version of a resource, starting with one for the oldest version."\
))

(defmethod get-resource-version ((resource versioned-resource) (index integer) &key (error\
-p t))
  (cond ((< index 1) (error "Illegal version index ~S" index))
        ((> index (resource-version-count resource))
         (when error-p
           (error "Version ~S of resource ~S does not exist." index resource))
         nil)
        (t (elt (resource-versions resource) (1- index)))))

(defgeneric get-newest-resource-version (resource &key error-p)
  (:documentation "Get the newest version of a resource."))

(defmethod get-newest-resource-version ((resource versioned-resource) &key (error-p t))
  (cond ((zerop (resource-version-count resource))
         (when error-p
           (error "Resource ~S has no versions.")))
        (t (car (last (resource-versions resource))))))

(defgeneric compute-print-name (resource)
  (:documentation "Compute and cache the print name for a resource."))

(defmethod compute-print-name ((resource resource))
  (setf (resource-print-name resource)
        (string-capitalize (resource-descriptor resource))))

(defmethod compute-print-name ((resource archive))
  (setf (resource-print-name resource)
        (concatenate 'string (string-capitalize (resource-descriptor resource))
                     " " (symbol-name (archive-name resource)))))

(defmethod compute-print-name ((resource link))
  (setf (resource-print-name resource) "Link"))

(defmethod compute-print-name ((resource collection))
  (setf (resource-print-name resource) (concatenate 'string "Collection " (collection-titl\
e resource))))
```

```
(defgeneric make-text-resource (archive text &key class save-p)
  (:documentation "Create a new text resource."))

(defmethod make-text-resource ((archive archive) (text string) &key (class 'text) (save-p \
t))
  (let ((new-text (make-persistent-instance class :string text :archive archive)))
    (compute-print-name new-text)
    (when save-p
      (save-object new-text))
    new-text))

(defgeneric make-http-resource (archive uri-string &key class save-p)
  (:documentation "Create a new reference to a HTTP resource."))

(defmethod make-http-resource ((archive archive) (uri-string string) &key (class 'http-res\
ource) (save-p t))
  (let ((new-http (make-persistent-instance class :http-identifier uri-string :archive arc\
hive)))
    (compute-print-name new-http)
    (when save-p
      (save-object new-http))
    new-http))

;;;---------------------------------------------------------------------
;;;
;;; LINKS
;;;

(defgeneric register-link (link &key save-p)
  (:documentation "Register a fully initialized link with its corresponding two resources.
   Returns the three objects."))

(defmethod register-link ((link link) &key (save-p t))
  (let ((from (link-from link))
        (to (link-to link)))
    (push link (from-links from))
    (push link (to-links to))
    (cond (save-p (save-objects (list link from to)))
          (t (list link from to)))))

(defgeneric filter-from-links (resource type)
  (:documentation "Return the list of resource from links that satisfy TYPE."))

(defmethod filter-from-links ((resource linkable-object-mixin) (type symbol))
  (loop for link in (from-links resource)
        when (typep link type)
        collect link))

(defgeneric filter-to-links (resource type)
  (:documentation "Return the list of resource to links that satisfy TYPE."))

(defmethod filter-to-links ((resource linkable-object-mixin) (type symbol))
  (loop for link in (to-links resource)
        when (typep link type)
        collect link))

(defgeneric link-objects (class from to &key save-p args)
  (:documentation "Create and register a link between two objects. Returns the three objec\
ts, link first."))

(defmethod link-objects ((class symbol) (from linkable-object-mixin) (to linkable-object-m\
ixin)
                         &key (save-p t) args)
  (let ((new-link (apply 'make-persistent-instance
                         class
                         :from from
                         :to to
                         :archive (resource-archive from)
```

```
                        args)))
    ; Should do this before link creation using a prototype instance.
    (unless (and (typep from (link-from-type new-link))
                 (typep to (link-to-type new-link)))
      (error "Incorrect object type for link ~S." new-link))
    (compute-print-name new-link)
    (register-link new-link :save-p save-p)))

(defgeneric annotate-resource (resource annotation author &key summary save-p)
  (:documentation "Associate an annotation with a resource."))

(defmethod annotate-resource ((resource archived-resource) (annotation archived-resource) \
author
                              &key summary
                              (save-p t))
  (link-objects 'annotation-link
                resource
                annotation
                :save-p save-p
                :args `(:author ,author ,@(when summary (list :summary summary)))))

(defmethod annotate-resource ((resource archived-resource) (annotation string) author
                              &key summary
                              (save-p t))
  "Annotation is saved as HTML text."
  (link-objects 'annotation-link resource
                (make-text-resource (resource-archive resource) annotation :class 'html-te\
xt)
                :save-p save-p
                :args `(:author ,author ,@(when summary (list :summary summary)))))

(defmethod annotate-resource ((resource identifier) annotation author &key summary (save-p\
 t))
  (annotate-resource (load-object resource) annotation author :summary summary :save-p sav\
e-p))

(defgeneric uri-annotate-resource (resource uri author &key summary save-p)
  (:documentation "Associate an HTTP resource with an archived resource."))

(defmethod uri-annotate-resource ((resource archived-resource) (uri string) author
                              &key summary
                              (save-p t))
  "Annotation is an HTTP identifier."
  (link-objects 'annotation-link resource
                (make-http-resource (resource-archive resource) uri)
                :save-p save-p
                :args `(:author ,author ,@(when summary (list :summary summary)))))

(defmethod uri-annotate-resource ((resource identifier) uri author &key summary (save-p t))
  (uri-annotate-resource (load-object resource) uri author :summary summary :save-p save-p\
))
```

11.4.2 code/cds/class.lisp

```
;;; -*- Mode: lisp; Syntax: ANSI-Common-Lisp; Base: 10; Package: CDS -*-
;;;
;;; Copyright 1998-99 Christopher R. Vincent (cvince@mit.edu)
;;; All Rights Reserved.
;;;
;;;---------------------------------------------------------------------
;;;
;;; CLASS DEFINITIONS
;;;

(in-package :cds)
```

```
;;;---------------------------------------------------------------------
;;;
;;; LINKS
;;;

(define-persistent-class linkable-object-mixin ()
  ((from-links
    :type list
    :allocation :persistent
    :initarg :from-links
    :initform nil
    :accessor from-links
    :documentation "Incoming links.")
   (to-links
    :type list
    :allocation :persistent
    :initarg :to-links
    :initform nil
    :accessor to-links
    :documentation "Outgoing links."))
  (:documentation "Mixin allowing objects to be linked."))

;;;---------------------------------------------------------------------
;;;
;;; LOCKS
;;;

(define-persistent-class lockable-object-mixin ()
  ((lock
    :initarg :lock
    :initform nil
    :accessor %get-lock
    :documentation "A simple lock for synchronization."))
  (:documentation "Mixin allowing objects to be locked."))

;;;---------------------------------------------------------------------
;;;
;;; RESOURCES
;;;

(define-persistent-class resource (linkable-object-mixin lockable-object-mixin)
  ((creation-time
    :allocation :persistent
    :initform (get-universal-time)
    :reader resource-creation-time)
   (creation-time-string
    :initarg :creation-time-string
    :initform nil
    :accessor %resource-creation-time-string)
   (descriptor
    :allocation :class
    :initform nil
    :reader resource-descriptor)
   (print-name
    :allocation :persistent
    :initarg :print-name
    :accessor resource-print-name))
  (:documentation "A basic resource."))

;;;---------------------------------------------------------------------
;;;
;;; ARCHIVES
;;;

(define-persistent-class archive (resource)
  ((name
    :type :keyword
    :allocation :persistent
    :initarg :name
```

```
    :accessor archive-name)
   (title
    :allocation :persistent
    :initarg :title
    :accessor archive-title)
   (index
    :allocation :persistent
    :initarg :index
    :accessor archive-index)
   (descriptor
    :allocation :class
    :initform "archive")
   ; Collections should be a more complex data structure, allowing multiple levels.
   (collections
    :allocation :persistent
    :initarg :collections
    :initform nil
    :accessor archive-collections
    :documentation "A list of collections in the archive."))
  (:documentation "An archive of resources."))

(define-persistent-class archived-resource (resource)
  ((archive
    :allocation :persistent
    :initarg :archive
    :accessor resource-archive))
  (:documentation "An archived resource."))

(define-persistent-class versioned-resource (archived-resource)
  ((versions
    :type list
    :allocation :persistent
    :initform nil
    :initarg :versions
    :accessor resource-versions)
   (version-count
    :type integer
    :allocation :persistent
    :initform 0
    :initarg :versions-count
    :accessor resource-version-count))
  (:documentation "An archived resource with an accompanying set of versions."))

(define-persistent-class resource-version (archived-resource)
  ((version-number
    :type integer
    :allocation :persistent
    :initarg :version-number
    :accessor resource-version-number)
   (descriptor
    :allocation :class
    :initform "version"))
  (:documentation "A version of a versioned-resource."))

(define-persistent-class collection (archived-resource)
  ((title
    :allocation :persistent
    :initarg :title
    :accessor collection-title)
   (elements
    :allocation :persistent
    :initarg :elements
    :accessor collection-elements)
   (element-type
    :allocation :class
    :initarg :element-type
    :initform 'archived-resource
    :accessor collection-element-type))
  (:documentation "A collection of archived resources."))
```

```
(define-persistent-class text (archived-resource)
  ((string
    :allocation :persistent
    :initarg :string
    :accessor text-string)
   (print-name
    :allocation :class
    :initform "Text"))
  (:documentation "An archived block of text."))

(define-persistent-class html-text (text)
  ((print-name
    :allocation :class
    :initform "HTML Text"))
  (:documentation "An archived block of HTML-formatted text."))

(define-persistent-class external-resource (archived-resource)
  () (:documentation "A resource that acts as a proxy for remote information."))

(define-persistent-class http-resource (external-resource)
  ((http-identifier
    :allocation :persistent
    :initarg :http-identifier
    :accessor http-resource-http-identifier
    :documentation "URI string")
   (descriptor
    :allocation :class
    :initform "HTTP resource"))
  (:documentation "An external resource that may be retrieved via HTTP."))

(define-persistent-class link (archived-resource)
  ((from
    :allocation :persistent
    :initarg :from
    :initform t
    :accessor link-from)
   (from-type
    :allocation :class
    :initarg :from-type
    :accessor link-from-type)
   (to
    :allocation :persistent
    :initarg :to
    :accessor link-to)
   (to-type
    :allocation :class
    :initarg :to-type
    :initform t
    :accessor link-to-type)
   (descriptor
    :allocation :class
    :initform "linked to"
    :reader link-descriptor)
   (inverse-descriptor
    :allocation :class
    :initform "linked by"
    :reader link-inverse-descriptor))
  (:documentation "A persistent link between two persistent objects."))

(define-persistent-class annotation-link (link)
  ((summary
    :allocation :persistent
    :initarg :summary
    :initform "Annotation."
    :accessor annotation-summary)
   (author
    :allocation :persistent
    :initarg :author
```

```
    :accessor annotation-author)
   (from-type
    :allocation :class
    :initform 'archived-resource)
   (to-type
    :allocation :class
    :initform 'archived-resource)
   (descriptor
    :allocation :class
    :initform "annotated by")
   (inverse-descriptor
    :allocation :class
    :initform "an annotation of"))
  (:documentation "An link annotating a resource."))

(define-persistent-class test-case-link (link)
  ((from-type
    :allocation :class
    :initform 'archived-resource)
   (to-type
    :allocation :class
    :initform 'archived-resource)
   (descriptor
    :allocation :class
    :initform "tested by")
   (inverse-descriptor
    :allocation :class
    :initform "a test case for"))
  (:documentation "A link associating a test case with a resource."))

;;;-------------------------------------------------------------------
;;;
;;; SOURCE CODE
;;;

(define-persistent-class code-archive (archive)
  ((descriptor
    :allocation :class
    :initform "code archive"))
  (:documentation "An archive of code resources."))

(define-persistent-class code-unit (archived-resource)
  ((implementations
    :type hash-table
    :allocation :persistent
    :initarg :implementations
    :initform (make-hash-table :test 'equal)
    :accessor code-unit-implementations
    :documentation "Maps implementation name to implementation.")
   (descriptor
    :allocation :class
    :initform "code unit"))
  (:documentation "A top-level code unit."))

(define-persistent-class implementation (versioned-resource)
  ((name
    :type string
    :allocation :persistent
    :initarg :name
    :accessor implementation-name
    :documentation "A name unique to the code-unit.")
   (code-unit
    :allocation :persistent
    :initarg :code-unit
    :accessor implementation-code-unit
    :documentation "The code-unit the implementation belongs to.")
   (descriptor
    :allocation :class
    :initform "implementation"))
```

```
  (:documentation "An implementation of a code-unit, may have multiple source code version\
s."))

(define-persistent-class source-code-version (resource-version)
  ((code-unit
    :allocation :persistent
    :initarg :code-unit
    :accessor version-code-unit
    :documentation "The code-unit the source-code belongs to.")
   (implementation
    :allocation :persistent
    :initarg :implementation
    :accessor version-implementation
    :documentation "The implementation a version belongs to."))
  (:documentation "One version of a code unit."))

(define-persistent-class code-collection (collection)
  ((element-type
    :allocation :class
    :initform 'source-code-version))
  (:documentation "A collection of source code versions."))

;;;---------------------------------------------------------------------
;;;
;;; LISP CODE
;;;

(define-persistent-class lisp-code-archive (code-archive)
  ((descriptor
    :allocation :class
    :initform "Lisp code archive"))
  (:documentation "An archive of Lisp code resources."))

(define-persistent-class lisp-code-unit (code-unit)
  ((package
    :type keyword
    :allocation :persistent
    :initarg :package
    :accessor code-unit-package)
   (descriptor
    :allocation :class
    :initform "Lisp code unit"))
  (:documentation "A Lisp code unit."))

(define-persistent-class lisp-named-code (lisp-code-unit)
  ((name
    :type string
    :allocation :persistent
    :initarg :name
    :accessor code-unit-name
    :documentation "This is canonically represented as downcase."))
  (:documentation "A Lisp code unit that has a name."))

(define-persistent-class lisp-anonymous-code (lisp-code-unit)
  () (:documentation "An unnamed Lisp code unit."))

(define-persistent-class lisp-code-container-mixin ()
  () (:documentation "Mixin for a Lisp form that contains other top-level forms."))

(define-persistent-class lisp-class (lisp-named-code)
  ((descriptor
    :allocation :class
    :initform "class"))
  (:documentation "A CLOS class definition."))

(define-persistent-class lisp-condition (lisp-named-code)
  ((descriptor
    :allocation :class
    :initform "condition"))
```

```
  (:documentation "A Lisp condition."))

(define-persistent-class lisp-generic-function (lisp-named-code)
  ((descriptor
    :allocation :class
    :initform "generic function"))
  (:documentation "A CLOS generic function definition."))

(define-persistent-class lisp-method (lisp-named-code)
  ((dispatch-args
    :allocation :persistent
    :initarg :dispatch-args
    :accessor method-dispatch-args
    :documentation "A list of class/type names corresponging to the arguments the
     generic function dispatches on.")
   (descriptor
    :allocation :class
    :initform "method"))
  (:documentation "A CLOS method definition."))

(define-persistent-class lisp-function (lisp-named-code)
  ((descriptor
    :allocation :class
    :initform "function"))
  (:documentation "A Lisp function definition."))

(define-persistent-class lisp-macro (lisp-named-code)
  ((descriptor
    :allocation :class
    :initform "macro"))
  (:documentation "A Lisp macro definition."))

(define-persistent-class lisp-parameter (lisp-named-code)
  ((descriptor
    :allocation :class
    :initform "parameter"))
  (:documentation "A Lisp parameter definition."))

(define-persistent-class lisp-setf (lisp-named-code)
  ((descriptor
    :allocation :class
    :initform "setf"))
  (:documentation "A Lisp setf definition."))

(define-persistent-class lisp-variable (lisp-named-code)
  ((descriptor
    :allocation :class
    :initform "variable"))
  (:documentation "A Lisp variable definition."))

(define-persistent-class lisp-constant (lisp-named-code)
  ((descriptor
    :allocation :class
    :initform "constant"))
  (:documentation "A Lisp constant definition."))

(define-persistent-class lisp-package (lisp-named-code)
  ((descriptor
    :allocation :class
    :initform "package"))
  (:documentation "A Lisp package definition."))

(define-persistent-class lisp-declamation (lisp-anonymous-code)
  ((descriptor
    :allocation :class
    :initform "declamation"))
  (:documentation "A Lisp top-level declaim statement."))

(define-persistent-class lisp-eval-when (lisp-anonymous-code lisp-code-container-mixin)
```

```
  ((descriptor
    :allocation :class
    :initform "eval-when"))
  (:documentation "A Lisp top-level eval-when statement."))

(define-persistent-class lisp-progn (lisp-anonymous-code lisp-code-container-mixin)
  ((descriptor
    :allocation :class
    :initform "progn"))
  (:documentation "A Lisp top-level progn statement."))

(define-persistent-class lisp-implementation (implementation)
  ((qualifiers
    :allocation :persistent
    :initarg :qualifiers
    :initform nil
    :accessor implementation-qualifiers
    :documentation "Compile-time qualifiers for this implementation, #+ syntax."))
  (:documentation "An implementation of a Lisp code unit."))

(define-persistent-class lisp-version (source-code-version)
  () (:documentation "A version of a Lisp implementation."))

(define-persistent-class lisp-source-code-version (lisp-version)
  ((source-code
    :allocation :persistent
    :initarg :source-code
    :accessor version-source-code))
  (:documentation "One version of a Lisp source code implementation."))

(define-persistent-class lisp-code-container-version (lisp-version)
  ((code-units
    :allocation :persistent
    :initarg :code-units
    :initform (make-array 0 :fill-pointer t :adjustable t)
    :accessor container-code-units)
   (situations
    :allocation :persistent
    :initarg :situations
    :initform t
    :accessor container-situations
    :documentation "Lisp evaluation situations, as defined for Lisp eval-when."))
  (:documentation "One version of a Lisp code container implementation."))

(define-persistent-class lisp-code-collection (code-collection)
  ((element-type
    :allocation :class
    :initform 'lisp-version))
  (:documentation "A collection of source code versions."))

(define-persistent-class lisp-annotation (archived-resource) ())

;;;-------------------------------------------------------------------
;;;
;;; LISP LINKS
;;;

(define-persistent-class lisp-link (link)
  () (:documentation "A Lisp link."))

; Keep class names short for losing Java/Macintosh combination.
(define-persistent-class lisp-version-ann-link (annotation-link)
  ((from-type
    :allocation :class
    :initform 'lisp-source-code-version)
   (start-idx
    :allocation :persistent
    :initarg :start-idx
    :accessor link-start-idx)
```

```
   (end-idx
    :allocation :persistent
    :initarg :end-idx
    :accessor link-end-idx))
  (:documentation "Associate a resource with a region of Lisp source code."))

(define-persistent-class lisp-method-of-link (lisp-link)
  ((from-type
    :allocation :class
    :initform 'lisp-method)
   (to-type
    :allocation :class
    :initform 'lisp-generic-function)
   (descriptor
    :allocation :class
    :initform "a method of")
   (inverse-descriptor
    :allocation :class
    :initform "the generic function for"))
  (:documentation "A method of a CLOS generic function."))

;;;-------------------------------------------------------------------
;;;
;;; SOURCE CODE PARSERS
;;;

(defclass parser ()
  ((buffer
    :initarg :buffer
    :accessor parser-buffer)
   (buffer-idx
    :initarg :buffer-idx
    :accessor parser-buffer-idx))
  (:documentation "Source code parser."))

(defclass lisp-parser (parser)
  ((archive
    :initarg :archive
    :accessor parser-archive)
   (package
    :initarg :package
    :accessor parser-package)
   (mode-line
    :initarg :mode-line
    :accessor parser-mode-line))
  (:documentation "A parser for Lisp code."))
```

11.4.3 code/cds/code-archive.lisp

```
;;; -*- Mode: lisp; Syntax: ANSI-Common-Lisp; Base: 10; Package: CDS -*-
;;;
;;; Copyright 1998-99 Christopher R. Vincent (cvince@mit.edu)
;;; All Rights Reserved.
;;;
;;;-------------------------------------------------------------------
;;;
;;; CODE ARCHIVE
;;;

(in-package :cds)

;;;-------------------------------------------------------------------
;;;
;;; ARCHIVE
;;;
```

```
;;;---------------------------------------------------------------------
;;;
;;; RESOURCES
;;;

(defgeneric name-string (instance)
  (:documentation "Compute a string name for a CDS object."))

(defmethod name-string ((instance code-archive))
  (symbol-name (archive-name instance)))

(defmethod name-string ((instance code-unit))
  "code-unit")

(defmethod name-string ((instance implementation))
  (implementation-name instance))

(defmethod name-string ((instance source-code-version))
  (format nil "Version ~D" (resource-version-number instance)))

(defgeneric register-implementation (code-unit implementation
                                               &key if-exists save-code-unit-p save-implem\
entation-p)
  (:documentation "Register an implementation for a code unit. IF-EXISTS may be :overwrite\
 or :error.
   If SAVE-CODE-UNIT-P implementation must be saved at least once before or during this op\
eration."))

(defmethod register-implementation ((code-unit code-unit) (implementation implementation)
                                    &key (if-exists :overwrite)
                                    (save-code-unit-p t)
                                    (save-implementation-p t))
  (let ((name (implementation-name implementation))
        (implementations (code-unit-implementations code-unit)))
    (when (and (eq :error if-exists)
               (gethash name implementations))
      (error "Implementation named ~A already registered in code-unit ~S" name code-unit))
    (setf (implementation-code-unit implementation) code-unit
          (gethash name implementations) implementation)
    (when save-implementation-p
      (save-object implementation))
    (when save-code-unit-p
      (save-object code-unit)))
  implementation)

(defgeneric get-implementation (name code-unit &key error-p)
  (:documentation "Get an implementation of a code-unit by name."))

(defmethod get-implementation ((name string) (code-unit code-unit) &key (error-p nil))
  (let ((implementation (gethash name (code-unit-implementations code-unit))))
    (cond (implementation)
          (error-p (error "Implementation named ~A not found in code-unit ~S" name code-un\
it))
          (t nil))))

(defgeneric map-implementations (function code-unit)
  (:documentation "Map over the implementations of a code-unit."))

(defmethod map-implementations ((function function) (code-unit code-unit))
  (maphash #'(lambda (key value)
               (declare (ignore key))
               (funcall function value))
           (code-unit-implementations code-unit))
  code-unit)
```

11.4.4 code/cds/java-serialization.lisp

```
;;; -*- Mode: lisp; Syntax: ANSI-Common-Lisp; Base: 10; Package: CDS -*-
;;;
;;; Copyright 1998-99 Christopher R. Vincent (cvince@mit.edu)
;;; All Rights Reserved.
;;;
;;;------------------------------------------------------------------
;;;
;;; SUPPORT FOR JAVA SERIALIZATION
;;;

(in-package :cds)

(defparameter *cds-java-serializable-classes*
  '(annotation-link
    html-text
    http-resource
    identifier
    lisp-class
    lisp-code-collection
    lisp-condition
    lisp-code-archive
    lisp-constant
    lisp-declamation
    lisp-eval-when
    lisp-function
    lisp-generic-function
    lisp-implementation
    lisp-macro
    lisp-method
    lisp-method-of-link
    lisp-package
    lisp-parameter
    lisp-progn
    lisp-setf
    lisp-source-code-version
    lisp-variable
    lisp-version-ann-link
    test-case-link
    text))

(defun generate-cds-java-sources (&optional (verbose t))
  ; Ensure everything is up to date.
  (flush-cached-class-mappings)
  ; Write out the source files.
  (generate-java-sources *cds-java-serializable-classes*
                         (pathname "cds:code;java;lisp;map;")
                         :verbose verbose))

(defmethod instance-for-serialization :before ((instance resource) recursive-p)
  "Ensure cached resource creation time string."
  (declare (ignore recursive-p))
  (resource-creation-time-string instance))

(defmethod instance-for-serialization ((instance lock) recursive-p)
  "Don't serialize locks."
  (declare (ignore recursive-p))
  nil)
```

11.4.5 code/cds/lisp-code.lisp

```
;;; -*- Mode: lisp; Syntax: ANSI-Common-Lisp; Base: 10; Package: CDS -*-
;;;
;;; Copyright 1998-99 Christopher R. Vincent (cvince@mit.edu)
;;; All Rights Reserved.
```

```
;;;
;;;---------------------------------------------------------------------
;;;
;;; LISP CODE
;;;

(in-package :cds)

;;;---------------------------------------------------------------------
;;;
;;; ARCHIVE
;;;

(defparameter *lisp-code-unit-type-alist*
  '((:class lisp-class :defclass :define-class :define-persistent-class)
    (:condition lisp-condition :define-condition)
    (:constant lisp-constant :defconstant :define-constant)
    (:declaim lisp-declamation :declaim)
    (:eval-when lisp-eval-when :eval-when)
    (:generic-function lisp-generic-function :defgeneric :define-generic)
    (:function lisp-function :defun :define)
    (:macro lisp-macro :defmacro :define-macro)
    (:method lisp-method :defmethod :define-method)
    (:package lisp-package :defpackage :define-package)
    (:parameter lisp-parameter :defparameter :define-parameter)
    (:progn lisp-progn :progn)
    (:setf lisp-setf :defsetf)
    (:variable lisp-variable :defvar :define-variable)
    (:other lisp-anonymous-code t))
  "Alist mapping Lisp code unit types to class name and keywords, where keyword is the fir\
st
   symbol in a top-level Lisp form.")

(defparameter *lisp-code-unit-keyword-table*
  (loop for spec in *lisp-code-unit-type-alist*
        for type = (first spec)
        for class-name = (second spec)
        with named-class = (find-class 'lisp-named-code)
        for named-p = (when (member named-class (class-precedence-list (find-class class-n\
ame))) t)
        with table = (make-hash-table)
        do (loop for keyword in (cddr spec)
                 do (setf (gethash keyword table) (list type class-name named-p)))
        finally (return table))
  "Maps Lisp code unit keywords to (type class-name named-p).  Used for parsing Lisp files\
.")

(defun code-unit-info-for-lisp-keyword (keyword)
  "Return list (type class-name named-p) for a Lisp keyword, for parsing Lisp files."
  (check-type keyword symbol)
  (or (gethash keyword *lisp-code-unit-keyword-table*)
      (gethash t *lisp-code-unit-keyword-table*)))

(defparameter *lisp-code-unit-class-name-table*
  (loop for spec in *lisp-code-unit-type-alist*
        for type = (first spec)
        for class-name = (second spec)
        with table = (make-hash-table)
        do (setf (gethash class-name table) type)
        finally (return table))
  "Maps Lisp code unit class-name to type.")

(defgeneric lisp-code-unit-type (code-unit)
  (:documentation "Return the type for a code-unit."))

(defmethod lisp-code-unit-type ((code-unit lisp-code-unit))
  (or (gethash (class-name (class-of code-unit)) *lisp-code-unit-class-name-table*)
      :other))
```

```
(defgeneric create-lisp-index-for-code-unit (code-unit)
  (:documentation "Create an appropriate sub-index for a code-unit."))

(defmethod create-lisp-index-for-code-unit ((code-unit lisp-named-code))
  (make-hash-table :test 'equal))

(defmethod create-lisp-index-for-code-unit ((code-unit lisp-anonymous-code))
  (make-array 0 :fill-pointer t :adjustable t))

(defgeneric mapcan-lisp-index (function sub-index)
  (:documentation "Map over code units in a Lisp sub-index, collect non-nil results."))

(defmethod mapcan-lisp-index ((function function) (sub-index hash-table))
  (loop for value being the hash-value of sub-index
        when value
        nconc (funcall function value)))

(defmethod mapcan-lisp-index ((function function) (sub-index vector))
  (loop for value across sub-index
        when value
        nconc (funcall function value)))

(defgeneric index-lisp-code-unit (code-unit sub-index)
  (:documentation "Add a Lisp code unit to its sub-index."))

(defmethod index-lisp-code-unit ((code-unit lisp-named-code) (sub-index hash-table))
  (setf (gethash (code-unit-name code-unit) sub-index)
        (object-identifier code-unit)))

(defmethod index-lisp-code-unit ((code-unit lisp-method) (sub-index hash-table))
  "Method names map to another hash-table, which maps from dispatch args list to lisp-meth\
od."
  (let* ((name-spec (code-unit-name code-unit))
         (method-table (or (gethash name-spec sub-index)
                           (setf (gethash name-spec sub-index) (make-hash-table :test 'equ\
al)))))
    (setf (gethash (method-dispatch-args code-unit) method-table)
          (object-identifier code-unit))))

(defmethod index-lisp-code-unit ((code-unit lisp-anonymous-code) (sub-index vector))
  (vector-push-extend (object-identifier code-unit) sub-index))

(defmethod make-index-for-archive ((archive lisp-code-archive))
  "Don't build actual sub-indexes until needed."
  nil)

(defmethod index-resource ((code-unit lisp-code-unit) (archive lisp-code-archive)
                           &key (type (lisp-code-unit-type code-unit)))
  (let ((index (archive-index archive)))
    (let ((sub-index (assoc type index)))
      (cond (sub-index (setq sub-index (cdr sub-index)))
            (t (setq sub-index (create-lisp-index-for-code-unit code-unit))
               (setf (archive-index archive) (acons type sub-index index))))
      (index-lisp-code-unit code-unit sub-index))))

(defmethod find-indexed-resource ((archive lisp-code-archive) (name-spec cons)
                                  &key (error-p nil) type dispatch-args)
  "Find a Lisp code-unit by (package . name), supply type to narrow search. name should be\
 downcase."
  (flet ((collect-methods (table)
           (cond (dispatch-args
                  (let ((id (gethash dispatch-args table)))
                    (when id (list (load-object id)))))
                 (t (loop for id being the hash-value of table
                          collect (load-object id))))))
    (declare (inline collect-methods))
    (cond  ((cond (type (let* ((sub-index (assoc type (archive-index archive)))
                               (entry (when sub-index (gethash name-spec (cdr sub-index)))\
))
```

```
                          (when entry
                            (case type
                              ; table of methods
                              (:method (collect-methods entry))
                              ; identifier for code-unit
                              (t (list (load-object entry)))))))
                    (t (loop for (type . sub-index) in (archive-index archive)
                             for found =
                             (when (typep sub-index 'hash-table)
                               (let ((entry (gethash name-spec sub-index)))
                                 (when entry
                                   (case type
                                     (:method (collect-methods entry))      ; table of metho\
ds
                                     (t (list (load-object entry)))))))     ; identifier for\
 code-unit
                             when found append found))))
              (error-p (error "Code unit not found for name ~S, supplied type ~S" name-spec t\
ype))
              (t nil))))

(defmethod find-indexed-resource ((archive lisp-code-archive) (name-spec null)
                                  &key (error-p nil) type dispatch-args)
  "Find all Lisp code-units of a specific type."
    (unless type
      (error "A code unit type must be supplied when name-spec is null."))
    (labels ((collect-index (id-or-methods)
               (etypecase id-or-methods
                 (pcs:identifier (list (load-object id-or-methods)))
                 (hash-table (cond (dispatch-args
                                    (when-bind (id (gethash dispatch-args id-or-methods))
                                      (list (load-object id))))
                                   (t (loop for id being the hash-value of id-or-methods
                                            collect (load-object id))))))))
      (let ((sub-index (when-bind (a (assoc type (archive-index archive))) (cdr a))))
        (when sub-index
          (or (cond ((mapcan-lisp-index #'collect-index sub-index))
                    (error-p
                     (error "No code units of type ~S found in archive." type))
                    (t nil)))))))

(defgeneric map-lisp-index (function sub-index)
  (:documentation "Map over a sub-index. Function accepts name and identifier."))

(defmethod map-lisp-index ((function function) (sub-index hash-table))
  (maphash #'(lambda (key value)
               (etypecase value
                 (identifier (funcall function key value))
                 (hash-table (map-lisp-index #'(lambda (sub-key sub-value)
                                                 (declare (ignore sub-key))
                                                 (funcall function key sub-value))
                                             value))))
           sub-index))

(defmethod map-lisp-index ((function function) (sub-index vector))
  (map nil #'(lambda (id)
               (funcall function nil id))
       sub-index))

(defmethod map-indexed-resources ((function function) (archive lisp-code-archive) &rest qu\
alifiers)
  "The qualifiers may be a type keyword or NIL for all resources."
  (cond (qualifiers
         (let* ((type (car qualifiers))
                (sub-index-assoc (assoc type (archive-index archive))))
           (when sub-index-assoc
             (map-lisp-index function (cdr sub-index-assoc)))))
        (t (loop for (nil . sub-index) in (archive-index archive)
                 do (map-lisp-index function sub-index)))))
```

```
(defmethod map-named-resources ((function function) (archive lisp-code-archive))
  "Map over all indexed resources that have a unique name."
  (loop for (nil . sub-index) in (archive-index archive)
        when (typep sub-index 'hash-table)
        do (map-lisp-index function sub-index)))

(defun make-lisp-code-archive (name title &key (class 'lisp-code-archive) (if-exists :erro\
r))
  "Create, save and register a new Lisp code archive.
   IF-EXISTS may be :overwrite or :error."
  (check-type name keyword)
  (when (and (eq :error if-exists)
             (get-archive name :error-p nil))
    (error "Archive ~S already exists." name))
  (let ((new-archive (make-persistent-instance class :name name :title title)))
    (setf (archive-index new-archive) (make-index-for-archive new-archive))
    (compute-print-name new-archive)
    (save-object new-archive)
    (register-archive new-archive)
    new-archive))

;;;-------------------------------------------------------------------
;;;
;;; EXPORTING INDEX DATA
;;;

(defgeneric export-named-resource-index (archive)
  (:documentation "Export an index of named resources for archive."))

(defmethod export-named-resource-index ((archive lisp-code-archive))
  (let ((package-table (make-hash-table :test #'equal)))
    (flet ((process-entry (name-spec id)
             (let* ((package (symbol-name (car name-spec)))
                    (name (cdr name-spec))
                    (name-table (gethash package package-table))
                    id-vect)
               (unless name-table
                 (setf name-table (make-hash-table :test #'equal)
                       (gethash package package-table) name-table))
               (setf id-vect (gethash name name-table))
               (unless id-vect
                 (setf id-vect (make-array 0 :fill-pointer t :adjustable t)
                       (gethash name name-table) id-vect))
               (vector-push-extend id id-vect))))
      (map-named-resources #'process-entry archive))
    package-table))

(defmethod export-named-resource-index ((archive identifier))
  (export-named-resource-index (load-object archive)))

;;;-------------------------------------------------------------------
;;;
;;; RESOURCES
;;;

(defmethod print-lisp-resource-object ((code-unit lisp-named-code) (stream stream))
  (cond ((object-cached-p code-unit)
         (with-slots (name) code-unit
           (format stream "LISP;~A;~A;~A" (lisp-code-unit-type code-unit) (car name) (cdr \
name))))
        (t (format stream "LISP;~A;unknown;unkown" (lisp-code-unit-type code-unit)))))

(defmethod print-lisp-resource-object ((code-unit lisp-code-unit) (stream stream))
  (cond ((object-cached-p code-unit)
         (with-slots (package) code-unit
           (format stream "LISP;~A;~A;anonymous" (lisp-code-unit-type code-unit) package)))
        (t (format stream "LISP;~A;unkown;anonymous" (lisp-code-unit-type code-unit)))))
```

```
(defmethod print-lisp-resource-object ((implementation lisp-implementation) (stream stream\
))
  (cond ((object-cached-p implementation)
         (with-slots (code-unit name) implementation
           (print-lisp-resource-object code-unit stream)
           (format stream ";~A" name)))
        (t (write-string "LISP;unkown;unknown;unkown;unkown" stream))))

(defmethod print-lisp-resource-object ((version lisp-version) (stream stream))
  (cond ((object-cached-p version)
         (with-slots (implementation version-number) version
           (print-lisp-resource-object implementation stream)
           (format stream ";~D" version-number)))
        (t (write-string "LISP;unkown;unknown;unkown;unkown;unkown" stream))))

(defmethod print-object ((instance lisp-code-unit) stream)
  (ignore-errors
   (print-unreadable-object (instance stream :type nil :identity t)
     (print-lisp-resource-object instance stream))))

(defmethod print-object ((instance lisp-implementation) stream)
  (ignore-errors
   (print-unreadable-object (instance stream :type nil :identity t)
     (print-lisp-resource-object instance stream))))

(defmethod print-object ((instance lisp-version) stream)
  (ignore-errors
   (print-unreadable-object (instance stream :type nil :identity t)
     (print-lisp-resource-object instance stream))))

(defparameter *lisp-default-implementation-name* "Default"
  "Name used for the default implementation of a Lisp code unit.")

(defmethod compute-print-name ((resource lisp-named-code))
  (destructuring-bind (package . name) (code-unit-name resource)
    (setf (resource-print-name resource)
          (concatenate 'string (string-capitalize (resource-descriptor resource))
                       " " (string-downcase (symbol-name package)) ":" name))))

(defun dispatch-args-to-string (args pkg)
  (labels ((write-spec (spec stream)
             (cond ((eq :keyword (car spec))
                    (write-char #\: stream)
                    (write-string (cdr spec) stream))
                   ((eq pkg (car spec))
                    (write-string (cdr spec) stream))
                   (t (format stream "~A:~A" (string-downcase (symbol-name (car spec))) (c\
dr spec))))))
    (with-output-to-string (stream)
      (write-char #\( stream)
      (when (first args)
        (write-char #\: stream)
        (write-string (cdr (first args)) stream)
        (write-char #\Space stream))
      (loop for arg in (rest args)
            for count upfrom 1
            with last = (length (rest args))
            when (eq t arg)
            do (write-char #\t stream)
            else when (eq :eql (car arg))
            do (write-string "(eql " stream)
            (write-spec (second arg) stream)
            (write-char #\) stream)
            else
            do (write-spec arg stream)
            unless (= count last)
            do (write-char #\Space stream))
      (write-char #\) stream))))
```

```
(defmethod compute-print-name ((resource lisp-method))
  (destructuring-bind (package . name) (code-unit-name resource)
    (setf (resource-print-name resource)
          (concatenate 'string (string-capitalize (resource-descriptor resource))
                       " " (string-downcase (symbol-name package)) ":" name
                       (dispatch-args-to-string (method-dispatch-args resource)
                                                (code-unit-package resource))))))

(defmethod compute-print-name ((resource lisp-implementation))
  (setf (resource-print-name resource)
        (concatenate 'string (resource-print-name (implementation-code-unit resource))
                     ", " (implementation-name resource))))

(defmethod compute-print-name ((resource lisp-source-code-version))
  (setf (resource-print-name resource)
        (format nil "~A v~D" (resource-print-name (version-implementation resource))
                (resource-version-number resource))))

(defun make-lisp-code-unit (archive package
                                    &key (class 'lisp-anonymous-code)
                                    name
                                    dispatch-args)
  "Create a new Lisp code unit. Must be registered using register-resource."
  (check-type package keyword)
  (check-type archive code-archive)
  (let ((code-unit (make-persistent-instance class
                                             :archive archive
                                             :package package)))
    (when name
      (setf (code-unit-name code-unit) name))
    (when dispatch-args
      (setf (method-dispatch-args code-unit) dispatch-args))
    (compute-print-name code-unit)
    code-unit))

(defun make-lisp-implementation (code-unit name
                                           &key (class 'lisp-implementation)
                                           qualifiers
                                           (save-implementation-p t)
                                           (save-code-unit-p t)
                                           (if-exists :replace))
  "Create and register a new implementation of a Lisp code unit.
   SAVE-IMPLEMENTATION-P is forced when SAVE-CODE-UNIT-P."
  (check-type code-unit lisp-code-unit)
  (check-type name string)
  (when (and (eq :error if-exists)
             (get-implementation name code-unit))
    (error "Implementation named ~S already exists for code unit ~S" name code-unit))
  (let ((new-implementation (make-persistent-instance class
                                                      :name name
                                                      :archive (resource-archive code-unit)
                                                      :code-unit code-unit
                                                      :qualifiers qualifiers)))
    (compute-print-name new-implementation)
    (register-implementation code-unit new-implementation
                             :save-implementation-p (or save-implementation-p save-code-un\
it-p)
                             :save-code-unit-p save-code-unit-p)
    new-implementation))

(defun make-lisp-source-code-version (implementation source-code
                                                     &key (class 'lisp-source-code-version)
                                                     (save-version-p t)
                                                     (save-implementation-p t))
  "Create and register a new version of a Lisp source code implementation.
   SAVE-VERSION-P is forced when SAVE-IMPLEMENTATION-P."
  (check-type implementation lisp-implementation)
  (check-type source-code string)
  (let ((new-version (make-persistent-instance class
```

```
                                                :archive (resource-archive implementation)
                                                :code-unit (implementation-code-unit implem\
entation)
                                                :implementation implementation
                                                :source-code source-code)))
    (register-new-version implementation new-version
                          :save-version-p save-implementation-p
                          :save-resource-p save-implementation-p)
    (compute-print-name new-version)
    (when save-version-p
      (save-object new-version))
    new-version))

(defun make-lisp-code-container-version (implementation situations code-units
                                                        &key (class 'lisp-code-container-v\
ersion)
                                                        (save-version-p t)
                                                        (save-implementation-p t))
  "Create and register a new version of a Lisp code container implementation.
   SAVE-VERSION-P is forced when SAVE-IMPLEMENTATION-P."
  (check-type implementation lisp-implementation)
  (check-type situations list)
  (check-type code-units vector)
  (let ((new-version (make-persistent-instance class
                                                :archive (resource-archive implementation)
                                                :code-unit (implementation-code-unit implem\
entation)
                                                :implementation implementation
                                                :situations situations
                                                :code-units code-units)))
    (register-new-version implementation new-version
                          :save-version-p save-implementation-p
                          :save-resource-p save-implementation-p)
    (compute-print-name new-version)
    (when save-version-p
      (save-object new-version))
    new-version))

(defun make-lisp-code-collection (archive title versions
                                          &key (class 'lisp-code-collection)
                                          (save-archive-p t)
                                          (save-collection-p t))
  "Create a collection of Lisp versions.
   SAVE-COLLECTION-P is forced when SAVE-ARCHIVE-P."
  (check-type archive lisp-code-archive)
  (check-type versions list)
  (check-type title string)
  (let ((new-collection (make-persistent-instance class
                                                  :archive archive
                                                  :title title
                                                  :elements versions)))
    (compute-print-name new-collection)
    (push new-collection (archive-collections archive))
    (when (or save-collection-p save-archive-p)
      (save-object new-collection))
    (when save-archive-p
      (save-object archive))
    new-collection))

;;;---------------------------------------------------------------------
;;;
;;; LINKS
;;;

(defgeneric ensure-method-of-links (archive &key save-p)
  (:documentation "Ensure that all CLOS methods are linked to their generic functions.
   Returns any modified objects."))

(defmethod ensure-method-of-links ((archive lisp-code-archive) &key (save-p t))
```

```
  (let ((methods (find-indexed-resource archive nil :type :method)))
    (loop for method in methods
          unless (filter-from-links method 'lisp-method-of-link)
          nconc (let* ((name (code-unit-name method))
                       (generic-functions (find-indexed-resource archive name :type :gener\
ic-function)))
                  (when generic-functions
                    (link-objects 'lisp-method-of-link method (car generic-functions) :sav\
e-p save-p))))))

#||

(defgeneric ensure-caller-links (archive &key save-p)
  (:documentation "Ensure that all Lisp caller links are installed."))

(defun get-code-units-called (version)
  "This is a temporary measure, shouldn't really depend on packages and on calling the Lis\
p reader."
  (let* ((source-code (version-source-code version))
         (package (code-unit-package (version-code-unit version)))
         (source-list (let ((*package* (or (find-package package)
                                           (make-package package))))
                        (read-from-string source-code)))
         (walker-output (do-code-walk source-list)))
    (inspect walker-output) (break)))

(defmethod ensure-caller-links ((archive lisp-code-archive) &key (save-p t))
  (let (modified)
    (labels ((install-version-callers (version)
               (let ((referenced (get-code-units-called version)))
                 (inspect referenced) (break)
                 ))
             (process-versions (implementation)
               (map-resource-versions #'install-version-callers implementation))
             (process-implementations (code-unit)
               (map-implementations #'process-versions code-unit))
             (process-code-units (type)
               (map-indexed-resources #'(lambda (name id)
                                          (declare (ignore name))
                                          (process-implementations (load-object id)))
                                      archive
                                      type)))
      (map nil #'process-code-units '(:function :method :macro :setf))
      (when save-p
        (save-objects modified)))))

||#

(defgeneric ensure-implicit-links (archive &key save-p)
  (:documentation "Ensure that all implicit links are installed. Returns list of modified \
objects."))

(defmethod ensure-implicit-links ((archive lisp-code-archive) &key (save-p t))
  (let (modified)
    (flet ((accumulate-modified (object-list)
             (loop for obj in object-list
                   do (pushnew obj modified))))
      (accumulate-modified (ensure-method-of-links archive :save-p nil)))
    (cond (save-p (save-objects modified))
          (t modified))))

(defgeneric annotate-version (version annotation author &key summary start end save-p)
  (:documentation "Associate an annotation with a region of source code."))

(defmethod annotate-version ((version lisp-source-code-version) (annotation archived-resou\
rce) author
                             &key summary
                             (start 0)
                             (end (length (version-source-code version)))
```

```
                              (save-p t))
  (link-objects 'lisp-version-ann-link version annotation :save-p save-p
                :args `(,@(when summary (list :summary summary))
                        :author ,author
                        :start-idx ,start
                        :end-idx ,end)))

(defmethod annotate-version ((version lisp-source-code-version) (annotation string) author
                             &key summary
                             (start 0)
                             end
                             (save-p t))
  "Annotation is saved as HTML text."
  (unless end
    (setq end  (length (version-source-code version))))
  (link-objects 'lisp-version-ann-link
                version
                (make-text-resource (resource-archive version) annotation :class 'html-tex\
t)
                :save-p save-p :args `(,@(when summary (list :summary summary))
                                       :author ,author
                                       :start-idx ,start
                                       :end-idx ,end)))

(defmethod annotate-version ((version identifier) annotation author
                             &key summary
                             (start 0)
                             end
                             (save-p t))
  (annotate-version (load-object version) annotation author :summary summary
                    :start start :end end :save-p save-p))
```

11.4.6 code/cds/lisp-parser.lisp

```
;;; -*- Mode: lisp; Syntax: ANSI-Common-Lisp; Base: 10; Package: CDS -*-
;;;
;;; Copyright 1998-99 Christopher R. Vincent (cvince@mit.edu)
;;; All Rights Reserved.
;;;
;;;---------------------------------------------------------------------
;;;
;;; LISP CODE PARSING
;;;

;;; Parser for moving conventional source code into CDS.
;;; Assume that every form we parse in is the first verion of a code-unit implementation.

(in-package :cds)

(defun make-lisp-parser (resource)
  (declare (ignore resource))
  (make-instance 'lisp-parser
    :buffer (make-array 1024 :element-type 'character :fill-pointer t :adjustable t)))

(defun initialize-lisp-parser (resource parser)
  (declare (ignore resource))
  (with-slots (buffer buffer-idx package mode-line) parser
    (setf (fill-pointer buffer) 0
          buffer-idx 0
          package :cl-user
          mode-line nil)))

(defresource lisp-parser ()
  :constructor make-lisp-parser
  :initializer initialize-lisp-parser)
```

```
(defmacro with-lisp-parser ((parser-var) &body body)
  `(using-resource (,parser-var lisp-parser)
     ,@body))

(defparameter *linear-whitespace-chars* '(#\Space #\Tab))

(defun linear-whitespace-p (char)
  (member char *linear-whitespace-chars*))

(defun trim-whitespace (string)
  (string-trim *linear-whitespace-chars* string))

(defparameter *line-delimiters* '(#\Return #\Linefeed))

(defun line-delimiter-p (char)
  (member char *line-delimiters*))

(defparameter *whitespace-chars*
  (append *linear-whitespace-chars* *line-delimiters*))

(defun whitespace-p (char)
  (member char *whitespace-chars*))

(defparameter *lisp-symbol-delimiters*
  (append *whitespace-chars* '(#\) #\( #\")))

(defun lisp-symbol-delimiter-p (char)
  (member char *lisp-symbol-delimiters*))

(defparameter *lisp-mode-line-delimiter* "-*-")

(defun buffer-read-line (stream buffer)
  "Append one line from a stream to a buffer."
  (loop for c = (read-char stream nil nil)
        while c
        until (line-delimiter-p c)
        do (vector-push-extend c buffer)))

(defun read-delimited (input delimiter-test read-function peek-function)
  (let ((chars (loop for c = (funcall peek-function input)
                     while c
                     until (funcall delimiter-test c)
                     collect (funcall read-function input))))
    (unless (zerop (length chars))
      (coerce chars 'string))))

(defun read-sharp-sign-plus-form (input read-function peek-function)
  "Capture a representation of a #+ conditionalization from stream."
  ; clear whitespace
  (loop for c = (funcall peek-function input)
        while (and c (whitespace-p c))
        do (funcall read-function input))
  (let ((next (funcall peek-function input)))
    (case next
      (#\(
       (funcall read-function input)
       (loop for entry = (read-sharp-sign-plus-form input read-function peek-function)
             while entry collect entry
             finally (unless (eq #\) (funcall read-function input))
                       (error "Expected ~S character." #\))))))
      (t (let* ((token-name (read-delimited input #'lisp-symbol-delimiter-p read-function \
peek-function)))
           (cond (token-name (intern-keyword token-name))
                 (t nil)))))))

(defun read-lisp-form (input output read-function peek-function output-function)
  (let ((output-count 0)
        sharp-sign-conditional)
    (labels ((read-input-char () (funcall read-function input))
```

```
            (peek-input-char () (funcall peek-function input))
            (clear-input-line ()
              (loop for c = (read-input-char)
                    until (line-delimiter-p c)))
            (advance-reader-comment (ignore)
              (loop
                for c = (read-input-char)
                unless ignore do (output-char c)
                until (and (eq #\| c) (eq #\# (peek-input-char)))
                finally (read-input-char)
                (unless ignore (output-char #\#))))
            (output-char (c)
              (incf output-count)
              (funcall output-function c output)))
      (declare (inline read-input-char peek-input-char clear-input-line advance-reader-com\
ment output-char))
      (loop for c = (read-input-char)
            with paren-depth = 0
            with quote-p
            with comment-p
            while c
            do (case c
                 (#\#
                  (unless (or comment-p quote-p)
                    (let ((next (peek-input-char)))
                      (cond ((eq #\| next)
                             (read-input-char)
                             (unless (zerop paren-depth)
                               (output-char c)
                               (output-char next))
                             (advance-reader-comment (zerop paren-depth)))
                            ((and (zerop paren-depth) (eq #\+ next))
                             (read-input-char)
                             (setq sharp-sign-conditional (read-sharp-sign-plus-form input\
 read-function peek-function)))
                            ((and (zerop paren-depth) (eq #\- next))
                             (read-input-char)
                             (setq sharp-sign-conditional
                                   (let ((form (read-sharp-sign-plus-form input read-funct\
ion peek-function)))
                                     (etypecase form
                                       (cons (append (list :not) form))
                                       (atom (append (list :not) (list form)))))))
                            (t (output-char c))))))
                 (#\\
                  (output-char c)
                  (output-char (read-input-char)))
                 (#\"
                  (unless comment-p
                    (setq quote-p (not quote-p)))
                  (output-char c))
                 (#\;
                  (cond ((or quote-p comment-p) (output-char c))
                        ((zerop paren-depth) (clear-input-line))
                        (t (setq comment-p t) (output-char c))))
                 (#\(
                  (unless (or comment-p quote-p)
                    (incf paren-depth))
                  (unless (zerop paren-depth)
                    (output-char c)))
                 (#\)
                  (cond ((or comment-p quote-p)
                         (unless (zerop paren-depth)
                           (output-char c)))
                        ((zerop paren-depth) (return nil))
                        ((zerop (decf paren-depth))
                         (output-char c)
                         (cond ((zerop output-count)
                                (return nil))
```

```
                              (t (return (values t sharp-sign-conditional)))))
                        (t (output-char c))))
                  (#\Return
                   (unless (zerop paren-depth)
                     (when (eq #\Linefeed (peek-input-char))
                       (read-input-char))
                     (output-char #\Return)
                     (output-char #\Linefeed))
                   (when comment-p (setq comment-p nil)))
                  (#\Linefeed
                   (unless (zerop paren-depth)
                     (output-char #\Return)
                     (output-char #\Linefeed))
                   (when comment-p (setq comment-p nil)))
                  (t (when (> paren-depth 0)
                       (output-char c))))))))

(defun buffer-read-lisp-form (stream buffer)
  (flet ((read-function (s)
           (read-char s nil nil))
         (peek-function (s)
           (peek-char nil s nil nil))
         (output-function (c b)
           (vector-push-extend c b)))
    (read-lisp-form stream buffer #'read-function #'peek-function #'output-function)))

(defun scan-buffered-lisp-form (parser)
  "Returns the start index (or NIL for nothing captured) and conditional of the next buffe\
red Lisp form.
   This should not be called when buffer-idx is in the middle of a top-level comment."
  (let ((length 0)
        start)
    (flet ((read-function (p)
             (prog1
               (elt (parser-buffer p) (parser-buffer-idx p))
               (incf (parser-buffer-idx p))))
           (peek-function (p)
             (elt (parser-buffer p) (parser-buffer-idx p)))
           (output-function (c p)
             (declare (ignore c))
             (unless start
               (setq start (1- (parser-buffer-idx p))))
             (incf length)))
      (multiple-value-bind (capture-p conditional)
                           (read-lisp-form parser parser #'read-function #'peek-function #\
'output-function)
        (cond (capture-p (values start conditional))
              (t (values nil nil)))))))

(defun parse-lisp-mode-line (parser)
  "Return an alist of mode-line properties. If not a valid mode-line, return NIL and leave\
 buffer-idx unchanged."
  (with-slots (buffer buffer-idx) parser
    (flet ((parse-property (string &aux (delim (position #\: string)))
             (when delim
               (let* ((name (intern-keyword (trim-whitespace (subseq string 0 delim))))
                      (value-string (trim-whitespace (subseq string (1+ delim)))))
                 (cons name (case name
                              (:package
                               (intern-keyword
                                (cond ((eq #\( (elt value-string 0))
                                       (subseq value-string 1 (position-if #'linear-whites\
pace-p value-string)))
                                      (t value-string))))
                              ((:mode :syntax)
                               (intern-keyword value-string))
                              (:base (parse-integer value-string :junk-allowed t))
                              (t value-string)))))))
      (declare (inline parse-property))
```

```
      (let* ((delim1 (search *lisp-mode-line-delimiter*  buffer :start2 buffer-idx))
             (delim2 (when delim1 (search *lisp-mode-line-delimiter* buffer :start2 (+ 3 d\
elim1)))))
        (when (and delim1 delim2)
          (setf buffer-idx (+ delim1 (length *lisp-mode-line-delimiter*)))
          (loop with start = (+ delim1 (length *lisp-mode-line-delimiter*))
                for idx from start to delim2
                for c = (elt buffer idx)
                with prev-delim = start
                with property
                when (or (eq c #\;) (= idx delim2))
                do (setq property  (parse-property (subseq buffer prev-delim idx))
                         prev-delim (1+ idx))
                and when property collect property))))))

(defun parse-symbol-name (parser)
  "Read a symbol from parser buffer, ignoring whitespace.  Allows double-quotes."
  (with-slots (buffer buffer-idx package) parser
    (let* ((name-start (let ((x (position-if-not #'whitespace-p buffer :start buffer-idx)))
                         (if (eq (elt buffer x) #\") (1+ x) x)))
           (name-end (position-if #'lisp-symbol-delimiter-p buffer :start name-start))
           (colon1-idx (position #\: buffer :start name-start :end name-end))
           (colon2-idx (when colon1-idx (position #\: buffer :start (1+ colon1-idx) :end n\
ame-end))))
      (let ((symbol-spec
             (cond ((and colon1-idx (= colon1-idx name-start))  ; keyword
                    (cons :keyword
                          (string-downcase(coerce (subseq buffer (1+ (or colon2-idx colon1\
-idx)) name-end) 'string))))
                   ((and colon1-idx colon2-idx)
                    (cons (intern-keyword (coerce (subseq buffer name-start colon1-idx) 's\
tring))
                          (string-downcase (coerce (subseq buffer (1+ colon2-idx) name-end\
) 'string))))
                   (colon1-idx
                    (cons (intern-keyword (coerce (subseq buffer name-start colon1-idx) 's\
tring))
                          (string-downcase (coerce (subseq buffer (1+ colon1-idx) name-end\
) 'string))))
                   (t (cons package (coerce (subseq buffer name-start name-end) 'string)))\
)))
        (setf buffer-idx name-end)
        symbol-spec))))

(defun parse-method-dispatch-args (parser)
  "Buffer index should be at the qualifier or opening paren of the specialized-lambda-list\
."
  (with-slots (buffer buffer-idx package) parser
    (labels ((advance-buffer-if (function)
               (setf buffer-idx (position-if function buffer :start buffer-idx)))
             (advance-buffer-if-not (function)
               (setf buffer-idx (position-if-not function buffer :start buffer-idx))))
      (advance-buffer-if-not #'whitespace-p)
      (let ((qualifier (unless (eq #\( (elt buffer buffer-idx))
                         (prog1
                           (parse-symbol-name parser)
                           (advance-buffer-if-not #'whitespace-p)))))
        (incf buffer-idx) ; advance past opening paren
        (loop for char = (elt buffer (advance-buffer-if-not #'whitespace-p))
              until (member char '(#\& #\)))
              collect
              (case char
                (#\(
                 (incf buffer-idx)
                 (parse-symbol-name parser)
                 (prog1
                   (progn
                     (advance-buffer-if-not #'whitespace-p)
                     (case (elt buffer buffer-idx)
```

```
                      (#\(
                       (incf buffer-idx)
                       (advance-buffer-if-not #'whitespace-p)
                       ; skip eql
                       (advance-buffer-if #'whitespace-p)                                    \

                       (prog1
                         (list :eql (parse-symbol-name parser))
                         (incf buffer-idx)))  ; clear #\)
                      (t (parse-symbol-name parser))))
                  (incf buffer-idx)))  ; clear #\)
               (t (parse-symbol-name parser)
                  t))
             into result
             finally (return (cons qualifier result)))))))

(defun parse-eval-when-situations (parser)
  "Buffer index should be at the opening paren of the situation list."
  (with-slots (buffer buffer-idx) parser
    ; advance past opening paren
    (setf buffer-idx (1+ (position-if-not #'whitespace-p buffer :start buffer-idx)))
    (loop for c = (elt buffer (position-if-not #'whitespace-p buffer :start buffer-idx))
          until (eq #\) c)
          collect (intern-keyword (cdr (parse-symbol-name parser)))
          ; clear closing paren
          finally (incf buffer-idx))))

(defun implementation-name-for-conditional (conditional)
  (labels ((walk-conditional (c)
             (when c
               (etypecase c
                 (symbol (symbol-name c))
                 (integer c)    ; acl allows numbers
                 (cons (cons (walk-conditional (car c)) (walk-conditional (cdr c))))))))
    (string-downcase (format nil "~A" (walk-conditional conditional)))))

;; This is extra kludgy because of the :method case.  Worth generalizing?
(defun process-parser-buffer (parser conditional
                                     &key (start 0)
                                     (end (length (parser-buffer parser)))
                                     (verbose t)
                                     (ignore-duplicates-p nil)
                                     (depth 0))
  "Buffer index should be positioned at the opening paren of a form."
  (with-slots (buffer buffer-idx archive package) parser
    (setf buffer-idx (1+ start))  ; skip opening paren
    (let* ((keyword (intern-keyword (cdr (parse-symbol-name parser)))))
      ; When in-package is encounted, change package and exit.
      (when (eq :in-package keyword)
        (setf package (intern-keyword (cdr (parse-symbol-name parser))))
        (return-from process-parser-buffer))
      (destructuring-bind (type class-name named-p) (code-unit-info-for-lisp-keyword keywo\
rd)
        (let* ((name (when named-p
                       (let ((name-start (position-if-not #'whitespace-p buffer :start buf\
fer-idx))
                             name-end)
                         ; detect (defun (setf ... )
                         (cond ((eq #\( (elt buffer name-start))
                                (setq name-end (1+ (position #\) buffer :start name-start)\
))
                                (setf buffer-idx (1+ name-end))
                                (cons package (coerce (subseq buffer name-start name-end) \
'string)))  ; default package okay?
                               ; normal name
                               (t (parse-symbol-name parser))))))
               (dispatch-args (when (eq :method type) (parse-method-dispatch-args parser)))
               (code-unit (or (when name (when-bind (a (find-indexed-resource archive name
                                                                              :type type :\
```

```
dispatch-args dispatch-args))
                                        (car a)))
                              (make-lisp-code-unit archive package :class class-name :name\
 name :dispatch-args dispatch-args)))
             (implementation-name (or (when conditional (implementation-name-for-conditi\
onal conditional))
                                      *lisp-default-implementation-name*))
             (implementation (cond ((and ignore-duplicates-p
                                         (get-implementation implementation-name code-un\
it))
                                    ; RETURN with NIL if ignoring duplicate definition.
                                    (return-from process-parser-buffer nil))
                                   (t (make-lisp-implementation code-unit
                                                                implementation-name
                                                                :qualifiers conditional
                                                                :if-exists :error
                                                                :save-implementation-p n\
il
                                                                :save-code-unit-p nil))))
             version)
          (flet ((process-anonymous-container (situations)
                   (make-lisp-code-container-version
                    implementation
                    situations
                    (loop for (sub-start sub-conditional) = (multiple-value-list (scan-buf\
fered-lisp-form parser))
                          for sub-end = buffer-idx
                          with result = (make-array 0 :fill-pointer t :adjustable t)
                          while sub-start
                          do (vector-push-extend (process-parser-buffer parser (cond ((and\
 conditional sub-conditional)
                                                                                      (lis\
t :and conditional sub-conditional))
                                                                                     (cond\
itional)
                                                                                     (t su\
b-conditional))
                                                                 :verbose verbose
                                                                 :start sub-start
                                                                 :end sub-end
                                                                 :ignore-duplicates\
-p ignore-duplicates-p
                                                                 :depth (1+ depth))
                                                 result)
                          finally (return result))
                    :save-version-p nil
                    :save-implementation-p nil)))
            (case keyword
              (:progn (setq version (process-anonymous-container nil)))
              (:eval-when (setq version (process-anonymous-container (parse-eval-when-situ\
ations parser))))
              (t (setq version (make-lisp-source-code-version
                                implementation
                                (coerce (make-array (- end start) :initial-contents (subse\
q buffer start end)) 'string)
                                :save-version-p nil
                                :save-implementation-p nil)))))
          (setf buffer-idx end)
          (save-objects (list version implementation code-unit))
          (register-resource code-unit archive :save-resource-p nil :save-archive-p nil)
          (when verbose
            (fresh-line)
            (write-string ";   ")
            (loop repeat depth do (write-string "   "))
            (write version))
          code-unit)))))

;; Should capture order, declaration scope information as part of a configuration.
(defun parse-lisp-file (pathname archive
```

```
                                &key implied-conditional
                                (verbose t)
                                (ignore-duplicates-p nil)
                                (save-archive-p t))
  "Parse a Lisp file, registering top-level forms in archive.
   Assume that each form encountered should be a new code unit implementation.
   Setting IGNORE-DUPLICATES-P to non-NIL will cause anonymous forms to be captured again."
  (check-type pathname pathname)
  (check-type archive lisp-code-archive)
  (with-lisp-parser (parser)
    (setf (parser-archive parser) archive)
    (with-slots (mode-line package buffer buffer-idx) parser
      (with-open-file (stream pathname
                              :direction :input
                              :if-does-not-exist :error)
        (when verbose
          (format t "~%; Parsing ~S." pathname))
        (when (eq #\; (peek-char t stream))
          (buffer-read-line stream buffer)
          (setf mode-line (parse-lisp-mode-line parser))
          ;; Set package using mode-line.
          (let ((mode-line-pkg-assoc (assoc :package mode-line)))
            (when mode-line-pkg-assoc
              (setf package (cdr mode-line-pkg-assoc))))
          (setf (fill-pointer buffer) 0
                buffer-idx 0))
        ;; Loop over top-level Lisp forms.
        (let ((count 0))
          (loop for (capture-p conditional) = (multiple-value-list (buffer-read-lisp-form \
stream buffer))
                while capture-p
                do (restart-case
                     (process-parser-buffer parser (cond ((and implied-conditional conditi\
onal)
                                                          (list :and implied-conditional c\
onditional))
                                                         (conditional conditional)
                                                         (t implied-conditional))
                                            :ignore-duplicates-p ignore-duplicates-p
                                            :verbose nil)
                     (skip-form () :report "Skip parsing form."))
                   (setf (fill-pointer buffer) 0
                         buffer-idx 0)
                   (incf count))
          (when save-archive-p
            (save-object archive))
          count)))))
```

11.4.7 code/cds/package.lisp

```
;;; -*- Mode: lisp; Syntax: ANSI-Common-Lisp; Base: 10; Package: cl-user; -*-
;;;
;;; Copyright 1998-99 Christopher R. Vincent (cvince@mit.edu)
;;; All Rights Reserved.
;;;
;;;---------------------------------------------------------------------
;;;
;;; SOURCE CODE SERVER
;;;

(in-package :cl-user)

(defpackage collaborative-development-server
  (:nicknames cds)
  (:use future-common-lisp cds-util pcs jos)
  (:export
```

```
  "ARCHIVE-CREATION-TIME"
  "ARCHIVE-NAME"
  "ARCHIVE-TITLE"
  "CODE-ARCHIVE"
  "CODE-ARCHIVE-COUNT"
  "CODE-UNIT"
  "CODE-UNIT-NAME"
  "FIND-INDEXED-RESOURCE"
  "GET-ARCHIVE"
  "IMPLEMENTATION"
  "IMPLEMENTATION-NAME"
  "MAP-ARCHIVES"
  "MAP-CODE-UNITS"
  "MAP-IMPLEMENTATIONS"
  "MAP-RESOURCE-VERSIONS"
  "NAME-STRING"
  "RESOURCE-ARCHIVE"
  "RESOURCE-CREATION-TIME"
  "RESOURCE-VERSION-NUMBER"
  "SOURCE-CODE-SOURCE-CODE"
  "SOURCE-CODE-VERSION"
  "WITH-RESOURCE-LOCKED"
  "WITH-RESOURCES-LOCKED"))
```

11.4.8 code/cds/test.lisp

```
;;; -*- Mode: lisp; Syntax: ANSI-Common-Lisp; Base: 10; Package: CDS -*-
;;;
;;; Copyright 1998-99 Christopher R. Vincent (cvince@mit.edu)
;;; All Rights Reserved.
;;;
;;;---------------------------------------------------------------------
;;;
;;; CDS TEST
;;;

(in-package :cds)

(defmacro do-cds-test ((string) &rest body)
  `(progn
     (format t "~%; Testing ~A." ,string)
     (unless (progn ,@body)
       (error "CDS test failed: ~A" ,string))
     t))

;; Careful, this test function both bashes the PCS instance cache and pollutes the databas\
e.
(defun test-cds (&aux archive code-unit1 imp1 imp2 code-unit2)
  archive imp2 ;ignorable
  (format t "~%; Test CDS start.")
  (do-cds-test ("new lisp-code-archive creation")
               (and (setq archive (make-lisp-code-archive :test-lisp-archive))
                    (eq archive (get-archive :test-lisp-archive))))
  (do-cds-test ("new lisp-code-unit creation")
               (and (setq code-unit1 (make-lisp-code-unit archive :package1 :class 'lisp-f\
unction :name "unit1"))
                    (setq code-unit2 (make-lisp-code-unit archive :package2 :class 'lisp-c\
lass :name "UNIT2"))
                    (make-lisp-code-unit archive :package1 :class 'lisp-eval-when)
                    (make-lisp-code-unit archive :package1)
                    (eq code-unit1 (find-indexed-resource archive (cons :package1 "unit1")\
 :type :function))
                    (eq code-unit2 (find-indexed-resource archive (cons :package2 "unit2")\
))))
  (do-cds-test ("new lisp-implementation creation")
               (and (setq imp1 (make-lisp-implementation code-unit1 "imp1"))
```

```
                    (setq imp2 (make-lisp-implementation code-unit1 "imp2"))))
  (do-cds-test ("new lisp-source-code-version creation")
               (and (make-lisp-source-code-version
                     imp1
                     "(defun unit1 (x) (1+ x))"
                     :documentation "An incrementor function.")
                    (make-lisp-source-code-version
                     imp1
                     "(defun unit1 (x) (+ 1 x))"
                     :documentation "An incrementor function.")
                    (make-lisp-source-code-version
                     imp1
                     "(defun unit1 (x) (- (+ 2 x) 1))"
                     :documentation "An incrementor function.")))
  (do-cds-test ("load archive after purging archive table and PCS instance cache")
               (progn
                 (purge-archive-table)
                 (pcs::purge-instance-cache)
                 (and (setq archive (get-archive :test-lisp-archive))
                      (setq code-unit1 (find-indexed-resource archive (cons :package1 "uni\
t1")))
                      (setq code-unit2 (find-indexed-resource archive (cons :package2 "uni\
t2") :type :class))
                      (eq archive (resource-archive code-unit1))
                      (eq archive (resource-archive code-unit2))
                      (string-equal "unit1" (code-unit-name code-unit1))
                      (string-equal "unit2" (code-unit-name code-unit2))
                      (eq :package1 (code-unit-package code-unit1))
                      (eq :package2 (code-unit-package code-unit2))
                      (setq imp1 (get-implementation "imp1" code-unit1))
                      (setq imp2 (get-implementation "imp2" code-unit1))
                      (string-equal "imp1" (implementation-name imp1))
                      (string-equal "(defun unit1 (x) (+ 1 x))"
                                    (source-code-source-code (get-resource-version imp1 2)\
)))))
  (format t "~%; Test CDS done.")
  t)
```

11.5 Network Interface

11.5.1 code/interface/object-server.lisp

```
;;; -*- Mode: lisp; Syntax: ANSI-Common-Lisp; Base: 10; Package: HTTP -*-
;;;
;;; Copyright 1998-99 Christopher R. Vincent (cvince@mit.edu)
;;; All Rights Reserved.
;;;
;;;------------------------------------------------------------------
;;;
;;; JAVA OBJECT SERVER
;;;

(in-package :http)

(define-content-type-name :java-serialization-stream
  :application :x-java-serialization-stream)

(defmethod post-document ((url url:http-form)
                          (type (eql :application))
                          (subtype (eql :x-java-serialization-stream))
                          stream
                          &aux bytes objects function)
  "Read a sequence of Java objects from the stream, pass to response function."
  (flet ((handle-error (error)
```

```
          (typecase error
            (network-error nil)   ;Pass through network errors for logging higher up.
            (t (setf (server-status *server*) 500)
               (error 'error-handling-post-method :url url :server-error error
                      :headers (header-plist)
                      :format-string "POST Error computing form response for ~A."
                      :format-args (list (url:name-string url))
                      :stack-backtrace (when *stack-backtraces-in-bug-reports*
                                         (stack-backtrace-string error))))))
        (parse-objects ()
          (jos:with-java-serialization-input (stream)
            (loop while (< (jos:input-byte-count) bytes)
                  collect (jos:read-object stream)))))
    (declare (inline parse-objects))
    (cond ((and (setq bytes (get-header :content-length))
                (setq objects (parse-objects))
                (setq function (url:response-function url)))
           (handler-bind
             ((error #'handle-error))
             (funcall function url stream objects)))
          ((null bytes)
           (error 'content-length-required :url url
                  :format-string "No content-length header provided for ~A."
                  :format-args (list (url:name-string url))))
          ((null objects)
           (error 'bad-syntax-provided :url url
                  :format-string "No form values were returned for ~A."
                  :format-args (list (url:name-string url))))
          (t (error 'server-internal-error :url url
                    :format-string "No Response function was found for ~A."
                    :format-args (list (url:name-string url)))))))

(defmethod java-load-object ((id pcs:identifier))
  (pcs:load-object id))

(defmethod java-load-object ((ids vector))
  (loop for id across ids
        for idx upfrom 0
        with result = (make-array (length ids))
        do (setf (elt result idx) (pcs:load-object id))
        finally (return result)))

(defmethod java-call-function ((url http-url) stream objects)
  "Perform a remote function call for a Java client. Objects: (FUNCTION-NAME &rest args)"
  (destructuring-bind (function-name &rest args) objects
    (let ((result (apply function-name args)))
      (with-successful-response (stream :java-serialization-stream)
        (jos:with-java-serialization-output (stream)
          (cond ((and (eq function-name 'java-load-object)
                      (typep (car args) 'vector))
                 ; Special hack to compensate for bad performance, due to
                 ; broken Java implementation of persistent HTTP connections.
                 (let ((pcs:*max-java-serialization-depth* 1))
                   (declare (special pcs:*max-java-serialization-depth*))
                   (jos:write-object result stream)))
                (t (jos:write-object result stream))))
        ; Need to force output?
        (force-output stream)))))

(defmethod cds-archive-info ((url http-url) stream)
  "Return a vector of archive info: #(archive1-title archive1-id archive2-title archive2-i\
d ...)"
  (let ((result (make-array 0 :fill-pointer t :adjustable t)))
    (cds:map-archives #'(lambda (archive)
                          (vector-push-extend (cds:archive-title archive) result)
                          (vector-push-extend (pcs:object-identifier archive) result)))
    (with-successful-response (stream :java-serialization-stream)
      (jos:with-java-serialization-output (stream)
        (jos:write-object result stream))
```

```
      ; Need to force output?
      (force-output stream))))

(defun export-cds-interface ()
  (export-url #u"/cds/java/funcall"
              :html-form
              :response-function #'java-call-function
              :pathname "cds:html;wrong-interface.html"
              :public t)
  (export-url #u"/cds/java/archive-info"
              :computed
              :response-function #'cds-archive-info
              :public t))
```

11.6 Configuration

11.6.1 configuration/cl-http/exports.lisp

```
;;; -*- Mode: lisp; Syntax: ANSI-Common-Lisp; Base: 10; Package: HTTP -*-
;;;
;;; Copyright 1998-99 Christopher R. Vincent (cvince@mit.edu)
;;; All Rights Reserved.
;;;
;;;------------------------------------------------------------------
;;;
;;; EXAMPLE MEDIA EXPORTS
;;;

(in-package :http)

(export-cds-interface)

(export-url #u"/"
            :directory
            :recursive-p t
            :pathname "cds:html;example-media"
            :expiration '(:interval ,(* 15. 60.))
            :public t
            :language :en)
```

11.6.2 configuration/platform/mcl/sysdcl.lisp

```
;;; -*- Mode: lisp; Syntax: ANSI-Common-Lisp; Base: 10; Package: CL-USER -*-
;;;
;;; Copyright 1998-99 Christopher R. Vincent (cvince@mit.edu)
;;; All Rights Reserved.
;;;
;;;------------------------------------------------------------------
;;;
;;; COLLABORATIVE DEVELOPMENT SERVER
;;;

(in-package :cl-user)

(load (merge-pathnames "translations"  ccl:*loading-file-source-file*) :verbose t)

(note-system-definitions
  (:cds-util "cds:configuration;platform;mcl;cds-util;sysdcl")
  (:cds-java "cds:configuration;platform;mcl;java;sysdcl")
  (:pcs "cds:configuration;platform;mcl;pcs;sysdcl")
```

```
  (:cds-archive "cds:configuration;platform;mcl;cds;sysdcl")
  (:cds-interface "cds:configuration;platform;mcl;interface;sysdcl")
  )

(setq *cds-systems* '(:cds-util
                      :cds-java
                      :pcs
                      :cds-archive
                      :cds-interface))

;; Load the standard components
(load-standard-systems *cds-systems*)
```

11.6.3 configuration/platform/mcl/translations.lisp

```
;;; -*- Mode: lisp; Syntax: ANSI-Common-Lisp; Base: 10; Package: USER -*-
;;;
;;; Copyright 1998-99 Christopher R. Vincent (cvince@mit.edu)
;;; All Rights Reserved.
;;;
;;;------------------------------------------------------------------
;;;
;;; LOGICAL PATHNAME HOST

(defun cds-directory-name-string (mcl-directory)
  (let ((mcl-dir (translate-logical-pathname mcl-directory)))
    (directory-namestring (make-pathname :device (pathname-device mcl-dir)
 :directory (butlast (pathname-directory mcl-dir) 3)))))

(defparameter *cds-directory* (cds-directory-name-string ccl:*loading-file-source-file*))

(defun cds-pathname (pathname)
   (concatenate 'string *cds-directory* pathname))

(defun cds-rooted-pathname (pathname)
   (concatenate 'string (subseq *cds-directory* 0 (1+ (position #\: *cds-directory* :test \
#'eql))) pathname))

(setf (logical-pathname-translations "cds")
      `(("cds;*.*"          ,(cds-pathname "*.*"))
("root;**;*.*"       ,(cds-rooted-pathname "**:*.*"))
("**;*.*"            ,(cds-pathname "**:*.*"))))
```

11.6.4 configuration/platform/mcl/cds/sysdcl.lisp

```
;;; -*- Mode: lisp; Syntax: ANSI-Common-Lisp; Base: 10; Package: CL-USER -*-
;;;
;;; Copyright 1998-99 Christopher R. Vincent (cvince@mit.edu)
;;; All Rights Reserved.
;;;
;;;------------------------------------------------------------------
;;;
;;; ARCHIVE
;;;

(in-package :cl-user)

(define-system
  (cds-archive)
  ()
  "cds:code;cds;package"
  "cds:code;cds;class"
```

```
  "cds:code;cds;archive"
  "cds:code;cds;code-archive"
  "cds:code;cds;lisp-code"
  "cds:code;cds;lisp-parser"
  "cds:code;cds;java-serialization")
```

11.6.5 configuration/platform/mcl/cds-util/sysdcl.lisp

```
;;; -*- Mode: lisp; Syntax: ANSI-Common-Lisp; Base: 10; Package: CL-USER -*-
;;;
;;; Copyright 1998-99 Christopher R. Vincent (cvince@mit.edu)
;;; All Rights Reserved.
;;;
;;;------------------------------------------------------------------
;;;
;;; UTIL
;;;

(in-package :cl-user)

(define-system
  (cds-util)
  ()
  "cds:code;cds-util;package"
  "cds:code;platform;mcl;cds-util;package"
  "cds:code;cds-util;util"
  "cds:code;platform;mcl;cds-util;weak-hash-table")
```

11.6.6 configuration/platform/mcl/cl-http/cl-http-init.lisp

```
;;; -*- Mode: lisp; Syntax: ANSI-Common-Lisp; Base: 10; Package: HTTP -*-
;;;
;;; Copyright 1998-99 Christopher R. Vincent (cvince@mit.edu)
;;; All Rights Reserved.
;;;
;;;------------------------------------------------------------------
;;;
;;; MCL INIT FILE
;;;

(in-package :http)

(cond ((string-equal (local-host-domain-name) "127.0.0.3")
       (load "http:mcl;examples;configuration-appletalk" :verbose t))
      (t (load "http:examples;configuration" :verbose t)))

(load "cds:configuration;cl-http;exports.lisp" :verbose t)

(enable-http-service)
```

11.6.7 configuration/platform/mcl/interface/sysdcl.lisp

```
;;; -*- Mode: lisp; Syntax: ANSI-Common-Lisp; Base: 10; Package: CL-USER -*-
;;;
;;; Copyright 1998-99 Christopher R. Vincent (cvince@mit.edu)
;;; All Rights Reserved.
;;;
;;;------------------------------------------------------------------
```

```
;;;
;;; HTTP
;;;

(in-package :cl-user)

(define-system
  (cds-interface)
  ()
  "cds:code;interface;object-server")
```

11.6.8 configuration/platform/mcl/java/sysdcl.lisp

```
;;; -*- Mode: lisp; Syntax: ANSI-Common-Lisp; Base: 10; Package: CL-USER -*-
;;;
;;; Copyright 1998-99 Christopher R. Vincent (cvince@mit.edu)
;;; All Rights Reserved.
;;;
;;;---------------------------------------------------------------------
;;;
;;; JAVA SUPPORT
;;;

(in-package :cl-user)

(define-system
  (cds-java)
  ()
  "cds:code;java;package"
  "cds:code;java;serialization")
```

11.6.9 configuration/platform/mcl/pcs/configuration.lisp

```
;;; -*- Mode: lisp; Syntax: ANSI-Common-Lisp; Base: 10; Package: PCS -*-
;;;
;;; Copyright 1998-99 Christopher R. Vincent (cvince@mit.edu)
;;; All Rights Reserved.
;;;
;;;---------------------------------------------------------------------
;;;
;;; PERSISTENT CLASS STORAGE CONFIGURATION
;;;

(in-package :pcs)

;;;---------------------------------------------------------------------
;;;
;;; SERIAL-NUMBERS
;;;

; Initialize the portable serial-number generator.

(setf *serial-number-state-pathname* (pathname "cds:data;sn-state"))

;;;---------------------------------------------------------------------
;;;
;;; STORAGE-MECHANISMS
;;;

#|| Initialize portable filesystem storage mechanism.

(defparameter *primary-filesystem-storage-mechanism*
```

```
  (allocate-storage-mechanism 'PRIMARY
                              'filesystem-storage-mechanism
                              :pathname (pathname "cds:data;pcs;")))

(setf *supported-storage-mechanisms* (list *primary-filesystem-storage-mechanism*))

(setf *default-storage-mechanism* *primary-filesystem-storage-mechanism*)

||#

; Initialize portable log-based storage mechanism.

(defvar *primary-log-based-storage-mechanism*
  (allocate-storage-mechanism 'PRIMARY
                              'log-based-storage-mechanism
                              :pathname (pathname "cds:data;instance-log")))

(setf *supported-storage-mechanisms* (list *primary-log-based-storage-mechanism*))

(setf *default-storage-mechanism* *primary-log-based-storage-mechanism*)
```

11.6.10 configuration/platform/mcl/pcs/sysdcl.lisp

```
;;; -*- Mode: lisp; Syntax: ANSI-Common-Lisp; Base: 10; Package: CL-USER -*-
;;;
;;; Copyright 1998-99 Christopher R. Vincent (cvince@mit.edu)
;;; All Rights Reserved.
;;;
;;;---------------------------------------------------------------------
;;;
;;; PERSISTENT CLASS STORAGE
;;;

(in-package :cl-user)

(define-system
  (pcs)
  ()
  "cds:code;pcs;package"
  "cds:code;pcs;class"
  "cds:code;pcs;serial-number"
  "cds:code;pcs;bin-dumper"
  "cds:code;pcs;storage-mechanism"
  "cds:code;pcs;identifier"
  "cds:code;pcs;class-info"
  "cds:code;pcs;system"
  ; "cds:code;pcs;filesystem-storage"
  "cds:code;pcs;log-based-storage"
  "cds:code;pcs;java-serialization"
  "cds:configuration;platform;mcl;pcs;configuration")
```

11.7 Java Client

11.7.1 code/java/AnnotationDialog.java

```
// Copyright 1998-99 Christopher R. Vincent (cvince@mit.edu)
// All Rights Reserved.

import java.awt.*;
import java.awt.event.*;
```

```
import java.util.*;
import javax.swing.*;
import javax.swing.event.*;
import lisp.lang.*;

public class AnnotationDialog extends JDialog implements ActionListener {

public static final int TYPE_URI = 0;
public static final int TYPE_HTML = 1;
public static final int TYPE_CANCELLED = 2;
static final int WIDTH = 600;

JButton commit_button, cancel_button;
public String annotation = null;
public int annotation_type = TYPE_HTML;
public String summary;
public String author = null;
JRadioButton uri_select, html_select;
JTextField uri_field, author_field, summary_field;
JTextArea annotation_html;
ButtonGroup select_group;

// Man, I should just write a layout tool for nested Box's!
public AnnotationDialog(JFrame owner) {
   super(owner, "New Annotation", true);
   setSize(WIDTH,500);
   Dimension horiz_space = new Dimension(Main.STANDARD_SPACING, 0);
   Dimension vert_space = new Dimension(0, Main.STANDARD_SPACING);
   Box author_box = new Box(BoxLayout.X_AXIS);
   author_box.add(new JLabel("Author:"));
   author_box.add(Box.createRigidArea(horiz_space));
   author_field = new JTextField();
   author_field.setMaximumSize(new Dimension(WIDTH / 2,
                               author_field.getMinimumSize().height));
   author_field.setMinimumSize(author_field.getMaximumSize());
   author_field.setPreferredSize(author_field.getMaximumSize());
   author_field.setText("aph@ai.mit.edu");
   author_box.add(author_field);
   author_box.add(Box.createHorizontalGlue());
   Box summary_box = new Box(BoxLayout.X_AXIS);
   summary_box.add(new JLabel("Summary:"));
   summary_box.add(Box.createRigidArea(horiz_space));
   summary_field = new JTextField();
   summary_field.setMaximumSize(new Dimension(Short.MAX_VALUE,
                                summary_field.getMinimumSize().height));
   summary_box.add(summary_field);
   Box type_box = new Box(BoxLayout.X_AXIS);
   type_box.add(new JLabel("Annotation Type:"));
   type_box.add(Box.createRigidArea(horiz_space));
   html_select = new JRadioButton("HTML Text");
   html_select.addActionListener(this);
   type_box.add(html_select);
   type_box.add(Box.createRigidArea(horiz_space));
   uri_select = new JRadioButton("Uniform Resource Identifier");
   uri_select.addActionListener(this);
   type_box.add(uri_select);
   type_box.add(Box.createHorizontalGlue());
   select_group = new ButtonGroup();
   select_group.add(uri_select);
   select_group.add(html_select);
   Box uri_box = new Box(BoxLayout.X_AXIS);
   uri_box.add(new JLabel("URI:"));
   uri_box.add(Box.createRigidArea(horiz_space));
   uri_field = new JTextField();
   uri_field.setMaximumSize(new Dimension(Short.MAX_VALUE,
                                          uri_field.getMinimumSize().height));
   uri_box.add(uri_field);
   annotation_html = new JTextArea();
   annotation_html.setEditable(true);
```

```
   JScrollPane html_pane = new JScrollPane(annotation_html,
                                           JScrollPane.VERTICAL_SCROLLBAR_ALWAYS,
                                           JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS);
   Box main_box = new Box(BoxLayout.Y_AXIS);
   main_box.add(summary_box);
   main_box.add(Box.createRigidArea(vert_space));
   main_box.add(author_box);
   main_box.add(Box.createRigidArea(vert_space));
   main_box.add(type_box);
   Box button_box = new Box(BoxLayout.X_AXIS);
   button_box.add(Box.createHorizontalGlue());
   cancel_button = new JButton("Cancel");
   cancel_button.addActionListener(this);
   button_box.add(cancel_button);
   button_box.add(Box.createRigidArea(horiz_space));
   commit_button = new JButton("Commit");
   commit_button.addActionListener(this);
   commit_button.setDefaultCapable(true);
   button_box.add(commit_button);
   main_box.add(Box.createRigidArea(vert_space));
   main_box.add(uri_box);
   main_box.add(Box.createRigidArea(vert_space));
   main_box.add(html_pane);
   main_box.add(Box.createRigidArea(vert_space));
   main_box.add(button_box);
   getContentPane().add("Center",
                        Main.createRigidPadding(Main.createBorderPanel(main_box)));
}

public void getAnnotation() {
   select_group.setSelected(html_select.getModel(), true);
   annotation_type = TYPE_HTML;
   summary_field.setText("Summary");
   uri_field.setText("http://server.net/annotation.html");
   uri_field.setEnabled(false);
   annotation_html.setText("<b>HTML</b> <font color=\"red\">Markup</font>.");
   annotation_html.setEnabled(true);
   setVisible(true);
}

public void actionPerformed(ActionEvent e) {
   JComponent source = (JComponent)e.getSource();
   if (source == commit_button) {
      summary = summary_field.getText();
      author = author_field.getText();
      switch (annotation_type) {
         case TYPE_HTML:
            annotation = annotation_html.getText();
            break;
         case TYPE_URI:
            annotation = uri_field.getText();
            break;
      }
      setVisible(false);
   }
   else if (source == cancel_button) {
      annotation_type = TYPE_CANCELLED;
      setVisible(false);
   }
   else if (source == html_select) {
      annotation_type = TYPE_HTML;
      uri_field.setEnabled(false);
      annotation_html.setEnabled(true);
      annotation_html.requestFocus();
   }
   else if (source == uri_select) {
      annotation_type = TYPE_URI;
      annotation_html.setEnabled(false);
      uri_field.setEnabled(true);
```

```
      uri_field.requestFocus();
   }
}

}
```

11.7.2 code/java/ArchiveInterface.java

```
// Copyright 1998-99 Christopher R. Vincent (cvince@mit.edu)
// All Rights Reserved.

import java.io.*;
import java.net.*;
import java.util.*;
import lisp.lang.*;
import lisp.map.*;

public class ArchiveInterface {

static final String ARCHIVE_INFO_NAME = "archive-info";
static final String FUNCALL_NAME = "funcall";
static final String SERIALIZATION_CONTENT_TYPE
   = "application/x-java-serialization-stream";

URL archive_info_url;
Vector args_vect;
Symbol find_resource_symbol;
Vector find_resource_vect;
URL funcall_url;
Keyword keyword_type, keyword_summary;
Symbol load_object_symbol;
Symbol archive_collections_symbol;
Symbol annotate_version_symbol;
Symbol annotate_resource_symbol;
Symbol get_resource_index_symbol;
Symbol uri_annotate_resource_symbol;
Hashtable object_cache;

public ArchiveInterface(URL base_url) {
   try {
      object_cache = new Hashtable();
      args_vect = new Vector();
      find_resource_vect = new Vector();
      funcall_url = new URL(base_url, FUNCALL_NAME);
      archive_info_url = new URL(base_url, ARCHIVE_INFO_NAME);
      keyword_type = new Keyword("TYPE");
      keyword_summary = new Keyword("SUMMARY");
      load_object_symbol = new Symbol("HTTP", "JAVA-LOAD-OBJECT");
      find_resource_symbol = new Symbol("CDS", "FIND-INDEXED-RESOURCE");
      archive_collections_symbol = new Symbol("CDS", "ARCHIVE-COLLECTIONS");
      annotate_version_symbol = new Symbol("CDS", "ANNOTATE-VERSION");
      annotate_resource_symbol = new Symbol("CDS", "ANNOTATE-RESOURCE");
      get_resource_index_symbol = new Symbol("CDS", "EXPORT-NAMED-RESOURCE-INDEX");
      uri_annotate_resource_symbol = new Symbol("CDS", "URI-ANNOTATE-RESOURCE");
   }
   catch (Exception e) {
      e.printStackTrace();
   }
}

private Object callLisp(URL location, Vector output) {
   URLConnection connection = null;
   try {
      connection = location.openConnection();
      if (output != null) {
         connection.setDoOutput(true);
```

```
            connection.setRequestProperty("content-type", SERIALIZATION_CONTENT_TYPE);
            ObjectOutputStream out = new ObjectOutputStream(connection.getOutputStream());
            for (Enumeration e = output.elements(); e.hasMoreElements();) {
               out.writeObject(e.nextElement());
            }
            out.flush();
            out.close();
         }
         ObjectInputStream in = new ObjectInputStream(connection.getInputStream());
         Object obj = in.readObject();
         in.close();
         return obj;
      }
      catch (Exception e) {
         if (connection != null) {
            try {
               System.err.println("HTTP Error: "
                                  +((HttpURLConnection)connection).getResponseMessage());
            }
            catch(Exception e2) {e2.printStackTrace(); }
         }
         e.printStackTrace();
         return null;
      }
   }

   private Object funcall(Symbol function_symbol, Vector args) {
      args.insertElementAt(function_symbol,0);
      return callLisp(funcall_url, args);
   }

   private synchronized Object callGetObject(Object id) {
      args_vect.removeAllElements();
      args_vect.addElement(id);
      return funcall(load_object_symbol, args_vect);
   }

   // This caching is more aggressive than for a real IDE.  Clients should register
   // with the server for modification updates.
   // Should use a weak table, but java.lang.ref isn't implemented for the Mac yet...
   public synchronized LispObject getObject(Pcs_Identifier identifier) {
      Number idnum = (Number)identifier.getSlotValue("serialNumber");
      Long serial_number = new Long(idnum.longValue());
      LispObject obj = (LispObject)object_cache.get(serial_number);
      if (obj == null) {
         obj = (LispObject)callGetObject(identifier);
         object_cache.put(serial_number, obj);
      }
      return obj;
   }

   public synchronized LispObject refreshObject(LispObject object) {
      Pcs_Identifier identifier = (Pcs_Identifier)object.getSlotValue("Pcs_identifier");
      Number idnum = (Number)identifier.getSlotValue("serialNumber");
      Long serial_number = new Long(idnum.longValue());
      LispObject obj = (LispObject)callGetObject(identifier);
      object_cache.put(serial_number, obj);
      return obj;
   }

   public synchronized LispObject[] getObjects(Vector identifiers) {
      Vector uncached_vect = new Vector();
      LispObject[] result = new LispObject[identifiers.size()];
      for (int i = 0; i < result.length; i++) {
         Pcs_Identifier id = (Pcs_Identifier)identifiers.elementAt(i);
         Number idnum = (Number)id.getSlotValue("serialNumber");
         Long serial_number = new Long(idnum.longValue());
         LispObject obj = (LispObject)object_cache.get(serial_number);
         if (obj == null) {
```

```
            result[i] = null;
            uncached_vect.addElement(id);
         }
         else {
            result[i] = obj;
         }
      }
      if (uncached_vect.size() == 0) {
         return result;
      }
      Object[] uncached = new Object[uncached_vect.size()];
      uncached_vect.copyInto(uncached);
      Object[] downloaded = (Object[])callGetObject(uncached);
      int downloaded_idx = 0;
      for (int i = 0; i < result.length; i++) {
         if (result[i] == null) {
            Pcs_Identifier id = (Pcs_Identifier)uncached[downloaded_idx];
            Number idnum = (Number)id.getSlotValue("serialNumber");
            Long serial_number = new Long(idnum.longValue());
            LispObject obj = (LispObject)downloaded[downloaded_idx];
            result[i] = obj;
            object_cache.put(serial_number, obj);
            downloaded_idx++;
         }
      }
      return result;
   }

   public synchronized Object findResource(Pcs_Identifier archive_id,
                                           Keyword pkg,
                                           String name,
                                           Keyword type) {
      args_vect.removeAllElements();
      args_vect.addElement(archive_id);
      // CDS code-unit name specs are (package-keyword . name-string) pairs.
      if ((pkg != null) && (name != null)) {
         find_resource_vect.removeAllElements();
         find_resource_vect.addElement(pkg);
         // Ensure name is lower-case.
         find_resource_vect.addElement(name.toLowerCase());
         args_vect.addElement(find_resource_vect);
      }
      else {
         args_vect.addElement(null);
      }
      if (type != null) {
         args_vect.addElement(keyword_type);
         args_vect.addElement(type);
      }
      return funcall(find_resource_symbol, args_vect);
   }

   // Return an alternating array of archive-title strings and archive identifiers.
   public synchronized Object[] getArchiveInfo() {
      return (Object[])callLisp(archive_info_url, null);
   }

   public synchronized Vector getArchiveCollections(Pcs_Identifier archive_id) {
      args_vect.removeAllElements();
      args_vect.addElement(archive_id);
      return (Vector)funcall(archive_collections_symbol, args_vect);
   }

   public LispObject getSlotObject(LispObject obj, String slot_name) {
      return getObject((Pcs_Identifier)obj.getSlotValue(slot_name));
   }

   public synchronized void annotateResource(LispObject resource,
                                             AnnotationDialog annotation) {
```

```
   args_vect.removeAllElements();
   args_vect.addElement(resource.getSlotValue("Pcs_identifier"));
   args_vect.addElement(annotation.annotation);
   args_vect.addElement(annotation.author);
   args_vect.addElement(keyword_summary);
   args_vect.addElement(annotation.summary);
   switch (annotation.annotation_type) {
      case AnnotationDialog.TYPE_URI:
         funcall(uri_annotate_resource_symbol, args_vect);
         break;
      case AnnotationDialog.TYPE_HTML:
         if (resource instanceof Cds_LispSourceCodeVersion) {
            funcall(annotate_version_symbol, args_vect);
         }
         else {
            funcall(annotate_resource_symbol, args_vect);
         }
         break;
   }
}

public synchronized Hashtable getResourceIndex(Pcs_Identifier archive_id) {
   args_vect.removeAllElements();
   args_vect.addElement(archive_id);
   return (Hashtable)funcall(get_resource_index_symbol, args_vect);
}

public static String nameSpecToString(Vector pair, Keyword pkg) {
   Keyword name_pkg = (Keyword)pair.elementAt(0);
   if (name_pkg == pkg) {
      return (String)pair.elementAt(1);
   }
   else {
      return name_pkg.name.toLowerCase() + ":" + (String)pair.elementAt(1);
   }
}

public static String dispatchArgsToString(Vector dispatch_args, Keyword pkg) {
   StringBuffer buf = new StringBuffer();
   if (dispatch_args.elementAt(0) != null) {
      buf.append(dispatch_args.elementAt(0).toString());
   }
   buf.append("(");
   for (int i = 1; i < dispatch_args.size() - 1; i++) {
      Vector arg = (Vector)dispatch_args.elementAt(i);
      if (arg.elementAt(0) instanceof Vector) {  // name-type pair
         buf.append("(" + nameSpecToString((Vector)arg.elementAt(0), pkg) + " "
                    + nameSpecToString((Vector)arg.elementAt(1), pkg) + ")\n");
      }
      else {  // bare variable name
         buf.append(nameSpecToString(arg, pkg));
      }
      if (i != dispatch_args.size() -2) {
         buf.append(" ");
      }
   }
   buf.append(")");
   return buf.toString();
}

public static boolean identifiersEqual(Pcs_Identifier id1, Pcs_Identifier id2) {
   if ((id1 == null) || (id2 == null)) {
      return false;
   }
   Number idnum1 = (Number)id1.getSlotValue("serialNumber");
   Number idnum2 = (Number)id2.getSlotValue("serialNumber");
   return idnum1.longValue() == idnum2.longValue();
}
```

```
public static boolean resourcesEqual(LispObject obj1, LispObject obj2) {
   Pcs_Identifier id1 = (Pcs_Identifier)obj1.getSlotValue("Pcs_identifier");
   Pcs_Identifier id2 = (Pcs_Identifier)obj2.getSlotValue("Pcs_identifier");
   return identifiersEqual(id1, id2);
}


}
```

11.7.3 code/java/CodeUnitPanel.java

```
// Copyright 1998-99 Christopher R. Vincent (cvince@mit.edu)
// All Rights Reserved.

import java.awt.*;
import java.awt.event.*;
import java.util.*;
import javax.swing.*;
import lisp.lang.*;
import lisp.map.*;

public class CodeUnitPanel extends JPanel {

static String NONE_AVAILABLE = "None Available";
static String NONE_SELECTED = "None Selected";
static String NOT_APPLICABLE = "Not Applicable";
static Color SELECTED_COLOR = Color.black;
static Color UNSELECTED_COLOR = new Color(75, 75, 75);

NavigatorPanel navigator;
LinkPanel link_panel;
Box code_unit_box, implementation_box, version_box;
JLabel code_unit_name, code_unit_package, version_time;
JComboBox implementation_choice, version_choice;
boolean listeners_enabled = false;
CodeBuffer buffer;
JScrollPane buffer_pane;
DefaultComboBoxModel empty_combobox_model;
Vector version_vect = null;
Hashtable implementation_table = null;
ArchiveInterface archive_interface;
Pcs_Identifier archive_id;
Hashtable resource_index;
Pcs_Identifier code_unit_id, implementation_id, version_id;

public CodeUnitPanel(NavigatorPanel navigator,
                     LinkPanel link_panel,
                     ArchiveInterface archive_interface,
                     Pcs_Identifier archive_id) {
   this.navigator = navigator;
   this.link_panel = link_panel;
   this.archive_interface = archive_interface;
   this.archive_id = archive_id;
   resource_index = archive_interface.getResourceIndex(archive_id);

   Box aux_box1;
   Dimension horiz_space = new Dimension(Main.STANDARD_SPACING, 0);
   Dimension vert_space = new Dimension(0, Main.STANDARD_SPACING);
   String[] none_available = {NONE_AVAILABLE};
   empty_combobox_model = new DefaultComboBoxModel(none_available);

   // code unit
   code_unit_box = new Box(BoxLayout.Y_AXIS);
   aux_box1 = new Box(BoxLayout.X_AXIS);
   aux_box1.add(new JLabel("Code Unit:"));
   aux_box1.add(Box.createRigidArea(horiz_space));
```

```
code_unit_name = new JLabel(NONE_SELECTED);
aux_box1.add(code_unit_name);
aux_box1.add(Box.createHorizontalGlue());
aux_box1.add(new JLabel("Package:"));
aux_box1.add(Box.createRigidArea(horiz_space));
code_unit_package = new JLabel(NONE_SELECTED);
aux_box1.add(code_unit_package);
code_unit_box.add(aux_box1);

// implementation
implementation_box = new Box(BoxLayout.X_AXIS);
implementation_box.add(new JLabel("Implementation:"));
implementation_box.add(Box.createRigidArea(horiz_space));
implementation_choice = new JComboBox();
implementation_choice.setEnabled(false);
implementation_choice.setMaximumSize(implementation_choice.getMinimumSize());
implementation_box.add(implementation_choice);
implementation_choice.addActionListener(new ActionListener() {
   public void actionPerformed(ActionEvent e) {
      if (listeners_enabled) {
         JComboBox cb = (JComboBox)e.getSource();
         changeImplementation((String)cb.getSelectedItem());
         showImplementationLinks();
      }
   }
});
implementation_box.add(Box.createHorizontalGlue());
implementation_choice.setModel(empty_combobox_model);

// version
version_box = new Box(BoxLayout.Y_AXIS);
aux_box1 = new Box(BoxLayout.X_AXIS);
aux_box1.add(new JLabel("Version:"));
aux_box1.add(Box.createRigidArea(horiz_space));
version_choice = new JComboBox();
version_choice.setEnabled(false);
version_choice.setMaximumSize(version_choice.getMinimumSize());
aux_box1.add(version_choice);
version_choice.addActionListener(new ActionListener() {
   public void actionPerformed(ActionEvent e) {
      if (listeners_enabled) {
         JComboBox cb = (JComboBox)e.getSource();
         changeVersion(Integer.parseInt((String)cb.getSelectedItem()));
         showVersionLinks();
      }
   }
});
aux_box1.add(Box.createHorizontalGlue());
aux_box1.add(new JLabel("Created:"));
aux_box1.add(Box.createRigidArea(horiz_space));
version_time = new JLabel();
aux_box1.add(version_time);
version_box.add(aux_box1);
version_choice.setModel(empty_combobox_model);

// buffer
buffer = new CodeBuffer(this, navigator.monospace_font);
buffer_pane = new JScrollPane(buffer,
                              JScrollPane.VERTICAL_SCROLLBAR_ALWAYS,
                              JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS);
buffer_pane.setPreferredSize(new Dimension(400, 400));

// top-level layout
Box main_box = new Box(BoxLayout.Y_AXIS);
main_box.add(Main.createBorderPanel(code_unit_box));
main_box.add(Box.createRigidArea(vert_space));
main_box.add(Main.createBorderPanel(implementation_box));
main_box.add(Box.createRigidArea(vert_space));
main_box.add(Main.createBorderPanel(version_box));
```

```
   main_box.add(Box.createRigidArea(vert_space));
   main_box.add(buffer_pane);
   setLayout(new GridLayout(1, 1));
   add(Main.createRigidPadding(main_box));
}

private void showCodeUnitLinks() {
   link_panel.showCodeUnit();
}

private void showImplementationLinks() {
   link_panel.showImplementation();
}

private void showVersionLinks() {
   link_panel.showVersion();
}

private void setDisplay(Pcs_Identifier new_code_unit_id,
                        Pcs_Identifier new_implementation_id,
                        Pcs_Identifier new_version_id) {
   listeners_enabled = false;
   if (!ArchiveInterface.identifiersEqual(new_code_unit_id, code_unit_id)) {
      code_unit_id = new_code_unit_id;
      LispObject code_unit = archive_interface.getObject(new_code_unit_id);
      if (code_unit instanceof Cds_LispMethod) {
         String print_name = (String)code_unit.getSlotValue("printName");
         int pos = print_name.indexOf('(');
         code_unit_name.setText(print_name.substring(0, pos));
      }
      else {
         code_unit_name.setText((String)code_unit.getSlotValue("printName"));
      }
      Keyword pkg = (Keyword)code_unit.getSlotValue("Cl_package");
      code_unit_package.setText(pkg.name.toLowerCase());
      implementation_table = (Hashtable)code_unit.getSlotValue("implementations");
      Enumeration names_enum = implementation_table.keys();
      String[] names = new String[implementation_table.size()];
      for (int i = 0; i < names.length; i++) {
         names[i] = (String)names_enum.nextElement();
      }
      implementation_choice.setModel(new DefaultComboBoxModel(names));
      implementation_choice.setEnabled(implementation_table.size() > 1);
      link_panel.setCodeUnit(code_unit);
   }
   if (!ArchiveInterface.identifiersEqual(new_implementation_id, implementation_id)) {
      implementation_id = new_implementation_id;
      LispObject implementation = archive_interface.getObject(new_implementation_id);
      implementation_choice.setSelectedItem((String)implementation.getSlotValue("name"));
      version_vect = (Vector)((Vector)implementation.getSlotValue("versions")).clone();
      version_vect.removeElementAt(version_vect.size() - 1);   // remove trailing null
      String[] names = new String[version_vect.size()];
      for (int i = 0; i < names.length; i++) {
         names[i] = Integer.toString(i+1);
      }
      version_choice.setModel(new DefaultComboBoxModel(names));
      version_choice.setEnabled(version_vect.size() > 1);
      link_panel.setImplementation(implementation);
   }
   version_id = new_version_id;
   LispObject version = archive_interface.getObject(new_version_id);
   int version_number = ((Number)version.getSlotValue("versionNumber")).intValue();
   version_choice.setSelectedIndex(version_number - 1);
   version_time.setText((String)version.getSlotValue("creationTimeString"));
   link_panel.setVersion(version);
   listeners_enabled = true;
}

private void setDisplay() {
```

```
   BufferElement selected_element = buffer.getSelectedElement();
   setDisplay(selected_element.code_unit_id,
              selected_element.implementation_id,
              selected_element.version_id);
}

public Pcs_Identifier defaultVersion(Pcs_Identifier implementation_id) {
   LispObject implementation = archive_interface.getObject(implementation_id);
   Vector version_vect = (Vector)implementation.getSlotValue("versions");
   // Last version before trailing null;
   return (Pcs_Identifier)version_vect.elementAt(version_vect.size() - 2);
}

public void changeImplementation(String name) {
   BufferElement selected_element = buffer.getSelectedElement();
   selected_element.implementation_id = (Pcs_Identifier)implementation_table.get(name);
   selected_element.version_id = defaultVersion(selected_element.implementation_id);
   selected_element.update();
   setDisplay();
}

public void changeVersion(int version_number) {
   BufferElement selected_element = buffer.getSelectedElement();
   selected_element.version_id = (Pcs_Identifier)version_vect.elementAt(version_number - 1\
);
   selected_element.update();
   setDisplay();
}

public synchronized void displayCollection(LispObject collection) {
   buffer.clear();
   Vector version_ids = (Vector)collection.getSlotValue("elements");
   version_ids = (Vector)version_ids.clone();
   version_ids.removeElementAt(version_ids.size() - 1);   // remove trailing null
   LispObject[] versions = archive_interface.getObjects(version_ids);

   // Performance hack, prefetch (cache) code units and implementations...
   Vector fetch = new Vector();
   for (int i = 0; i < versions.length; i++) {
      LispObject version = versions[i];
      fetch.addElement(version.getSlotValue("implementation"));
      fetch.addElement(version.getSlotValue("codeUnit"));
   }
   archive_interface.getObjects(fetch);

   for (int i = 0; i < versions.length; i++) {
      buffer.append(versions[i]);
   }
   buffer_pane.validate();
   buffer.selectElement(buffer.elementAt(0));
}

public synchronized void displayResource(LispObject resource) {
   Pcs_Identifier new_code_unit_id, new_implementation_id, new_version_id;
   // Kludgy way to determine resource type...
   if (resource instanceof Cds_LispSourceCodeVersion) {
      new_code_unit_id = (Pcs_Identifier)resource.getSlotValue("codeUnit");
      new_implementation_id = (Pcs_Identifier)resource.getSlotValue("implementation");
      new_version_id = (Pcs_Identifier)resource.getSlotValue("Pcs_identifier");
   }
   else if (resource instanceof Cds_LispImplementation) {
      new_code_unit_id = (Pcs_Identifier)resource.getSlotValue("codeUnit");
      new_implementation_id = (Pcs_Identifier)resource.getSlotValue("Pcs_identifier");
      new_version_id = null;
   }
   else {  // Assume it is a code unit.
      new_code_unit_id = (Pcs_Identifier)resource.getSlotValue("Pcs_identifier");
      new_implementation_id = null;
      new_version_id = null;
```

```
    }
    BufferElement element = buffer.findCodeUnit(new_code_unit_id);
    if (element == null) {
       element = buffer.append(resource);
       buffer_pane.validate();
    }
    else {
       if (new_implementation_id != null) {
          element.implementation_id = new_implementation_id;
          if (new_version_id != null) {
             element.version_id = new_version_id;
          }
          element.update();
       }
    }
    buffer.selectElement(element);
    buffer_pane.getViewport().setViewPosition(element.getLocation());
}

private void metaDot(Symbol symbol) {
    Hashtable name_table = (Hashtable)resource_index.get(symbol.pkg);
    if (name_table == null) {
       Main.beep();
       return;
    }
    Object[] choices = (Object[])name_table.get(symbol.name);
    if (choices == null) {
       Main.beep();
       return;
    }
    if (choices.length == 1) {
       displayResource(archive_interface.getObject((Pcs_Identifier)choices[0]));
       return;
    }
    // This should query the user for a choice of definition!
    displayResource(archive_interface.getObject((Pcs_Identifier)choices[0]));
}

static final char[] symbol_delim = {')', '(', ' ', ';', ',', '\'', '"', '\n'};

private boolean symbolDelimP(char c) {
    for (int i = 0; i < symbol_delim.length; i++) {
       if (c == symbol_delim[i]) {
          return true;
       }
    }
    return false;
}

private Symbol extractSymbol(Keyword default_pkg, String text, int position) {
    int start = 0;
    int end = text.length() - 1;
    // Find beginning of symbol.
    for (int i = position; i >= 0; i--) {
       if (symbolDelimP(text.charAt(i))) {
          if (i == position) {
             return null;
          }
          else {
             start = i + 1;
          }
          break;
       }
    }
    for (int i = position + 1; i < text.length() - 1; i++) {
       if (symbolDelimP(text.charAt(i))) {
          end = i;
          break;
       }
```

```
    }
    if (start == end) {
       return null;
    }
    String s = text.substring(start, end);
    int colon1_pos = s.indexOf(':');
    if (colon1_pos == -1) {
       return new Symbol(default_pkg.name, s);
    }
    else if (colon1_pos == 0) {
       return new Symbol("KEYWORD", s.substring(1));
    }
    int colon2_pos = s.indexOf(':', colon1_pos + 1);
    if (colon2_pos == -1) {
       colon2_pos = colon1_pos;
    }
    return new Symbol(s.substring(0, colon1_pos).toUpperCase(),
                      s.substring(colon2_pos + 1));
}

class CodeBuffer extends Box implements FocusListener, KeyListener {

    CodeUnitPanel panel;
    BufferElement selected_element = null;
    Font font;
    BufferElement dummy;

    public CodeBuffer(CodeUnitPanel panel, Font font) {
       super(BoxLayout.Y_AXIS);
       this.panel = panel;
       this.font = font;
       // Dummy element fills extra space in scroll pane.
       dummy = new BufferElement();
       dummy.setEditable(false);
       add(dummy);
    }

    public void selectElement(BufferElement element) {
       if (selected_element != null) {
          selected_element.deselect();
       }
       selected_element = element;
       element.select();
       panel.setDisplay(element.code_unit_id,
                        element.implementation_id,
                        element.version_id);
       element.requestFocus();
    }

    public BufferElement getSelectedElement() {
       return selected_element;
    }

    public BufferElement elementAt(int index) {
       return (BufferElement)getComponent(index);
    }

    public int length() {
       return getComponentCount() - 1;  // don't count dummy element
    }

    public void clear() {
       removeAll();
       add(dummy);
    }

    public BufferElement findCodeUnit(Pcs_Identifier code_unit_id) {
       Component[] elements = getComponents();
       if (elements.length > 1) {
```

```
      for (int i = 0; i < elements.length - 1; i++) {
         BufferElement element = (BufferElement)elements[i];
         if (ArchiveInterface.identifiersEqual(code_unit_id, element.code_unit_id)) {
            return element;
         }
      }
   }
   return null;
}

public BufferElement append(LispObject code_unit,
                            LispObject implementation,
                            LispObject version) {
   Pcs_Identifier implementation_id;
   if (implementation == null) {
      implementation_table = (Hashtable)code_unit.getSlotValue("implementations");
      implementation_id = (Pcs_Identifier)implementation_table.elements().nextElement();
   }
   else {
      implementation_id = (Pcs_Identifier)implementation.getSlotValue("Pcs_identifier");
   }
   Pcs_Identifier version_id;
   if (version == null) {
      version_id = defaultVersion(implementation_id);
   }
   else {
      version_id = (Pcs_Identifier)version.getSlotValue("Pcs_identifier");
   }
   Pcs_Identifier code_unit_id = (Pcs_Identifier)code_unit.getSlotValue("Pcs_identifier\
");
   BufferElement element = new BufferElement(code_unit_id, implementation_id,
                                             version_id, font);
   element.addFocusListener(this);
   element.addKeyListener(this);
   int pos = length();
   if (pos > 0) {
      BufferElement prev_element = (BufferElement)getComponent(pos-1);
      element.prev = prev_element;
      prev_element.next = element;
   }
   add(element, pos);
   invalidate();
   return element;
}

public BufferElement append(LispObject resource) {
   // Kludgy way to determine resource type...
   if (resource instanceof Cds_LispSourceCodeVersion) {
      return append(archive_interface.getSlotObject(resource, "codeUnit"),
                    archive_interface.getSlotObject(resource, "implementation"),
                    resource);
   }
   else if (resource instanceof Cds_LispImplementation) {
      return append(archive_interface.getSlotObject(resource, "codeUnit"),
                                                    resource, null);
   }
   else {  // Assume it is a code unit.
      return append(resource, null, null);
   }
}

public void focusGained(FocusEvent e) {
   BufferElement element = (BufferElement)e.getComponent();
   if (element != selected_element) {
      selectElement(element);
      showCodeUnitLinks();
   }
}
```

```
public void focusLost(FocusEvent e) {}

// Arrow keys can change buffer element selection.
private void handleArrowKey(int key_code, BufferElement element) {
   BufferElement new_element = null;
   int caret_pos = element.getCaretPosition();
   int line;
   boolean new_caret_start = true;
   switch (key_code) {
      case KeyEvent.VK_LEFT:
         if (caret_pos == 0) {
            new_element = element.prev;
            new_caret_start = false;
         }
         break;
      case KeyEvent.VK_RIGHT:
         int length = element.getDocument().getLength();
         if (caret_pos == length) {
            new_element = element.next;
         }
         break;
      case KeyEvent.VK_UP:
         try {
            line = element.getLineOfOffset(caret_pos);
         }
         catch(Exception e) {
            return;
         }
         if (line == 0) {
            new_element = element.prev;
            new_caret_start = false;
         }
         break;
      case KeyEvent.VK_DOWN:
         try {
            line = element.getLineOfOffset(caret_pos);
         }
         catch(Exception e) {
            return;
         }
         if (line == (element.getLineCount() - 1)) {
            new_element = element.next;
         }
         break;
   }
   if (new_element != null) {
      selectElement(new_element);
      new_element.setCaretPosition(new_caret_start ?
         0 : new_element.getDocument().getLength());
   }
}


public void keyPressed(KeyEvent e) {
   int code = e.getKeyCode();
   if ((code == KeyEvent.VK_LEFT) ||
       (code == KeyEvent.VK_RIGHT) ||
       (code == KeyEvent.VK_UP) ||
       (code == KeyEvent.VK_DOWN)) {
      handleArrowKey(code, (BufferElement)e.getComponent());
   }
   else if ((code == KeyEvent.VK_PERIOD)
            && ((e.getModifiers() & InputEvent.ALT_MASK) != 0)) {
      BufferElement element = (BufferElement)e.getComponent();
      LispObject code_unit = archive_interface.getObject(element.code_unit_id);
      Keyword default_pkg = (Keyword)code_unit.getSlotValue("Cl_package");
      Symbol symbol = extractSymbol(default_pkg,
                                    element.getText(),
                                    element.getCaretPosition());
```

```
            if (symbol == null) {
               Main.beep();
               return;
            }
            metaDot(symbol);
         }
      }

      public void keyTyped(KeyEvent e) {}

      public void keyReleased(KeyEvent e) {}

}

class BufferElement extends JTextArea {

      public Pcs_Identifier code_unit_id = null;
      public Pcs_Identifier implementation_id = null;
      public Pcs_Identifier version_id = null;
      public BufferElement prev = null;
      public BufferElement next = null;

      public BufferElement() {}

      public BufferElement(Pcs_Identifier code_unit_id,
                           Pcs_Identifier implementation_id,
                           Pcs_Identifier version_id,
                           Font font) {
         this.code_unit_id = code_unit_id;
         this.implementation_id = implementation_id;
         this.version_id = version_id;
         setFont(font);
         setForeground(UNSELECTED_COLOR);
         setMargin(new Insets(10,5,10,5));
         setMaximumSize(new Dimension(Short.MAX_VALUE, getMinimumSize().height));
         update();
      }

      public void update() {
         setText((String)(archive_interface.getObject(version_id)).getSlotValue("sourceCode")\
);
      }

      public void select() {
         setForeground(SELECTED_COLOR);
      }

      public void deselect() {
         setForeground(UNSELECTED_COLOR);
      }

}

}
```

11.7.4 code/java/ConfigurationFrame.java

```
// Copyright 1998-99 Christopher R. Vincent (cvince@mit.edu)
// All Rights Reserved.

import java.awt.*;
import java.awt.event.*;
import java.net.*;
import java.util.*;
import javax.swing.*;
import javax.swing.event.*;
```

```
import lisp.lang.*;
import lisp.map.Pcs_Identifier;

public class ConfigurationFrame extends JFrame
       implements ActionListener, ListSelectionListener {

static String DEFAULT_HOST = "motile.mit.edu";
static int DEFAULT_PORT = 80;
static String NEW = "Navigate";
static String UPDATE = "Update Archives";

JTextField server_name, server_port;
JButton update_button, new_button;
JList list;
IdentifierListModel model;
ArchiveInterface archive_interface = null;
Main main;
Pcs_Identifier selected = null;

public ConfigurationFrame(Main main) {
   super("Configure Navigator");
   this.main = main;

   Dimension horiz_space = new Dimension(Main.STANDARD_SPACING, 0);
   Dimension vert_space = new Dimension(0, Main.STANDARD_SPACING);

   // server
   Box server_box = new Box(BoxLayout.X_AXIS);
   server_box.add(new JLabel("Server:"));
   server_box.add(Box.createRigidArea(horiz_space));
   String saved_name = (String)main.getProperty("server_name");
   server_name = new JTextField((saved_name != null) ? saved_name : DEFAULT_HOST, 20);
   server_name.setMaximumSize(new Dimension(Short.MAX_VALUE,
                              server_name.getMinimumSize().height));
   server_box.add(server_name);
   server_box.add(Box.createRigidArea(horiz_space));
   server_box.add(new JLabel(":"));
   server_box.add(Box.createRigidArea(horiz_space));
   String saved_port = (String)main.getProperty("server_port");
   server_port = new JTextField((saved_port != null) ?
                                saved_port : Integer.toString(DEFAULT_PORT), 4);
   server_port.setMaximumSize(new Dimension(Short.MAX_VALUE,
                              server_port.getMinimumSize().height));
   server_box.add(server_port);

   // buttons
   Box button_box = new Box(BoxLayout.X_AXIS);
   update_button = new JButton(UPDATE);
   update_button.addActionListener(this);
   button_box.add(update_button);
   new_button = new JButton(NEW);
   new_button.setEnabled(false);
   new_button.addActionListener(this);
   button_box.add(Box.createHorizontalGlue());
   button_box.add(new_button);

   // archives
   list = new JList();
   list.setMaximumSize(new Dimension(Short.MAX_VALUE, Short.MAX_VALUE));
   list.addListSelectionListener(this);
   JScrollPane archives_pane = new JScrollPane(list);

   // top-levl layout
   Box main_box = new Box(BoxLayout.Y_AXIS);
   main_box.add(server_box);
   main_box.add(Box.createRigidArea(vert_space));
   main_box.add(button_box);
   main_box.add(Box.createRigidArea(vert_space));
   main_box.add(archives_pane);
```

```
   Container content = getContentPane();
   content.setLayout(new GridLayout(1, 1));
   content.add(Main.createRigidPadding(Main.createBorderPanel(main_box)));
   pack();
   setVisible(true);
}

public void actionPerformed(ActionEvent e) {
   String s = e.getActionCommand();
   if (s.equals(UPDATE)) {
      try {
         URL url = new URL("http://"+server_name.getText()+
                           ":"+server_port.getText()+"/cds/java/");
         archive_interface = new ArchiveInterface(url);
         model = new IdentifierListModel();
         Object[] archive_specs = archive_interface.getArchiveInfo();
         for(int i = 0; i < archive_specs.length; i+= 2) {
            model.append((String)archive_specs[i], (Pcs_Identifier)archive_specs[i+1]);
         }
         list.setModel(model);
      }
      catch (Exception err) {
         Main.beep();
         server_name.requestFocus();
         err.printStackTrace();
      }
   }
   else if (s.equals(NEW)) {
      main.setProperty("server_name", server_name.getText());
      main.setProperty("server_port", server_port.getText());
      main.saveProperties();
      new Navigator(archive_interface, selected, this);
      setVisible(false);
   }
}

public void valueChanged(ListSelectionEvent e) {
   if (e.getValueIsAdjusting())
      return;
   if (!list.isSelectionEmpty()) {
      new_button.setEnabled(true);
      selected = model.identifierAt(list.getSelectedIndex());
   }
}

}
```

11.7.5 code/java/HistoryFrame.java

```
// Copyright 1998-99 Christopher R. Vincent (cvince@mit.edu)
// All Rights Reserved.

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import lisp.lang.*;

public class HistoryFrame extends JFrame {

HistoryPanel panel;

public HistoryFrame(NavigatorPanel navigator) {
   super("Navigator History");
   panel = new HistoryPanel(navigator);
   setContentPane(panel);
}
```

```
public void appendHistory(LispObject resource) {
   panel.appendHistory(resource);
}

}
```

11.7.6 code/java/HistoryPanel.java

```
// Copyright 1998-99 Christopher R. Vincent (cvince@mit.edu)
// All Rights Reserved.

import java.awt.*;
import java.awt.event.*;
import java.util.Vector;
import javax.swing.*;
import javax.swing.event.*;
import lisp.lang.*;
import lisp.map.Pcs_Identifier;

// Need to synchronize the cell renderer.

public class HistoryPanel extends JPanel
                          implements ActionListener, ListSelectionListener {

static final int DEFAULT_MAX_LENGTH = 128;
static final String CLEAR = "Clear";

JList list;
NavigatorPanel navigator;
IdentifierListModel model;
JButton clear_button;

public HistoryPanel(NavigatorPanel navigator) {
   this.navigator = navigator;
   Dimension horiz_space = new Dimension(Main.STANDARD_SPACING, 0);
   Dimension vert_space = new Dimension(0, Main.STANDARD_SPACING);
   model = new IdentifierListModel(DEFAULT_MAX_LENGTH);
   clear_button = new JButton(CLEAR);
   clear_button.addActionListener(this);
   list = new JList(model);
   list.addListSelectionListener(this);
   list.setMaximumSize(new Dimension(Short.MAX_VALUE, Short.MAX_VALUE));
   Box aux_box1 = new Box(BoxLayout.Y_AXIS);
   Box aux_box3 = new Box(BoxLayout.X_AXIS);
   aux_box3.add(new JLabel("Resources Visited:"));
   aux_box3.add(Box.createHorizontalGlue());
   aux_box3.add(clear_button);
   aux_box1.add(aux_box3);
   aux_box1.add(Box.createRigidArea(vert_space));
   aux_box1.add(list);
   setLayout(new GridLayout(1,1));
   add(Main.createRigidPadding(Main.createBorderPanel(aux_box1)));
}

public void appendHistory(LispObject resource) {
   model.append(resource);
}

public void actionPerformed(ActionEvent e) {
   String s = e.getActionCommand();
   if (s.equals(CLEAR)) {
      model.clear();
   }
}
```

```
public void valueChanged(ListSelectionEvent e) {
   if (e.getValueIsAdjusting())
      return;
   if (!list.isSelectionEmpty()) {
      navigator.displayResource(model.identifierAt(list.getSelectedIndex()));
   }
}

}
```

11.7.7 code/java/IdentifierListModel.java

```
// Copyright 1998-99 Christopher R. Vincent (cvince@mit.edu)
// All Rights Reserved.

import java.awt.*;
import java.awt.event.*;
import java.net.*;
import java.util.*;
import javax.swing.*;
import lisp.lang.*;
import lisp.map.Pcs_Identifier;

public class IdentifierListModel extends AbstractListModel {

   int max_length;
   Vector names, ids;

   public IdentifierListModel(int max_length) {
      this.max_length = max_length;
      names = new Vector();
      ids = new Vector();
   }

   public IdentifierListModel() {
      this(Integer.MAX_VALUE);
   }

   public void append(String name, Pcs_Identifier id) {
      if (names.size() == max_length) {
         names.setSize(max_length - 1);
         ids.setSize(max_length - 1);
         fireIntervalRemoved(this, max_length - 1, max_length - 1);
      }
      names.insertElementAt(name, 0);
      ids.insertElementAt(id, 0);
      fireIntervalAdded(this, 0, 0);
   }

   public synchronized void append(LispObject resource) {
      append((String)resource.getSlotValue("printName"),
             (Pcs_Identifier)resource.getSlotValue("Pcs_identifier"));
   }

   public synchronized void clear() {
      int length = names.size();
      if (length > 0) {
         names.removeAllElements();
         ids.removeAllElements();
         fireIntervalRemoved(this, 0, length - 1);
      }
   }

   public synchronized int getSize() {
      return names.size();
   }
```

```
   public synchronized Object getElementAt(int index) {
      return names.elementAt(index);
   }

   public synchronized Pcs_Identifier identifierAt(int index) {
      return (Pcs_Identifier)ids.elementAt(index);
   }

}
```

11.7.8 code/java/LinkPanel.java

```
// Copyright 1998-99 Christopher R. Vincent (cvince@mit.edu)
// All Rights Reserved.

import java.awt.*;
import java.awt.event.*;
import java.util.*;
import javax.swing.*;
import javax.swing.event.*;
import javax.swing.text.*;
import javax.swing.tree.*;
import lisp.lang.*;
import lisp.map.*;

public class LinkPanel extends JPanel {

NavigatorPanel navigator;
ArchiveInterface archive_interface;
JTabbedPane tabbed_pane;
LinkDisplayBox code_unit_links, implementation_links, version_links;
AnnotationDialog annotation_dialog;

public LinkPanel(NavigatorPanel navigator, ArchiveInterface archive_interface) {
   this.navigator = navigator;
   this.archive_interface = archive_interface;
   annotation_dialog = new AnnotationDialog(navigator.frame);
   code_unit_links = new LinkDisplayBox();
   implementation_links = new LinkDisplayBox();
   version_links = new LinkDisplayBox();
   tabbed_pane = new JTabbedPane();
   tabbed_pane.addTab("  Code Unit  ", code_unit_links);
   tabbed_pane.addTab("  Implementation  ", implementation_links);
   tabbed_pane.addTab("  Version  ", version_links);
   setLayout(new GridLayout(1, 1));
   add(tabbed_pane);
}

public void setCodeUnit(LispObject resource) {
   code_unit_links.setLinks(resource);
}

public void setImplementation(LispObject resource) {
   implementation_links.setLinks(resource);
}

public void setVersion(LispObject resource) {
   version_links.setLinks(resource);
}

public void showCodeUnit() {
   tabbed_pane.setSelectedIndex(0);
}

public void showImplementation() {
```

```
   tabbed_pane.setSelectedIndex(1);
}

public void showVersion() {
   tabbed_pane.setSelectedIndex(2);
}

class LinkDisplayBox extends Box implements ActionListener, ListSelectionListener {

   JTree tree;
   JScrollPane tree_pane, text_pane, link_list_pane;
   JTextPane annotation_text;
   JList link_list;
   IdentifierListModel link_model;
   JButton new_button, respond_button;
   LispObject selected_annotation = null;
   LispObject resource = null;

   public LinkDisplayBox() {
      super(BoxLayout.Y_AXIS);
      Dimension horiz_space = new Dimension(Main.STANDARD_SPACING / 2, 0);
      Box link_label = new Box(BoxLayout.X_AXIS);
      link_label.add(new JLabel("General Relations:"));
      link_label.add(Box.createHorizontalGlue());
      link_model = new IdentifierListModel();
      link_list = new JList(link_model);
      link_list.addListSelectionListener(this);
      link_list_pane = new JScrollPane(link_list);
      Box link_box = new Box(BoxLayout.Y_AXIS);
      link_box.add(Main.createRigidPadding(link_label, Main.STANDARD_SPACING / 2));
      link_box.add(link_list_pane);
      tree = new JTree(new DefaultTreeModel(new DefaultMutableTreeNode()));
      tree.setRootVisible(false);
      tree.addTreeSelectionListener(new TreeSelectionListener() {
         public void valueChanged(TreeSelectionEvent e) {
            AnnotationTreeNode node =
               (AnnotationTreeNode)tree.getLastSelectedPathComponent();
            if (node == null) {
               return;
            }
            LispObject annotation = archive_interface.getSlotObject(node.link, "to");
            if (annotation instanceof Cds_HttpResource) {
               try {
                  annotation_text.setPage((String)annotation.getSlotValue("httpIdentifier"\
));
               }
               catch (Exception err) {
                  err.printStackTrace();
               }
            }
            else {
               annotation_text.setEditorKit(
                  annotation_text.getEditorKitForContentType("text/html"));
               // annotation_text.setContentType("text/html");
               String header = "<code>Author: "
                  +(String)node.link.getSlotValue("author") + "</code><br>"
                  + "<code>Date: "+(String)node.link.getSlotValue("creationTimeString")
                  + "</code><p>";
               annotation_text.setText(header + (String)annotation.getSlotValue("Cl_string\
"));
            }
            selected_annotation = node.link;
            respond_button.setEnabled(true);
         }
      });
      tree_pane = new JScrollPane(tree);
      annotation_text = new JTextPane();
      annotation_text.setEditable(false);
      text_pane = new JScrollPane(annotation_text);
```

```
   Box annotation_label = new Box(BoxLayout.X_AXIS);
   annotation_label.add(new JLabel("Resource Annotations:"));
   annotation_label.add(Box.createHorizontalGlue());
   annotation_label.add(Box.createRigidArea(horiz_space));
   new_button = new JButton(" New ");
   new_button.setBorder(BorderFactory.createEtchedBorder());
   new_button.setEnabled(false);
   new_button.addActionListener(this);
   annotation_label.add(new_button);
   respond_button = new JButton(" Respond ");
   respond_button.addActionListener(this);
   respond_button.setBorder(BorderFactory.createEtchedBorder());
   respond_button.setEnabled(false);
   annotation_label.add(Box.createRigidArea(horiz_space));
   annotation_label.add(respond_button);
   Box tree_box = new Box(BoxLayout.Y_AXIS);
   tree_box.add(Main.createRigidPadding(annotation_label, Main.STANDARD_SPACING / 2)); \

   tree_box.add(tree_pane);
   JSplitPane annotation_pane = new JSplitPane(JSplitPane.VERTICAL_SPLIT,
                                               tree_box, text_pane);
   annotation_pane.setDividerSize(Main.DIVIDER_SIZE);
   annotation_pane.setDividerLocation(200);
   JSplitPane main_pane = new JSplitPane(JSplitPane.VERTICAL_SPLIT,
                                         link_box, annotation_pane);
   main_pane.setDividerSize(Main.DIVIDER_SIZE);
   add(Main.createRigidPadding(main_pane, Main.DIVIDER_SIZE));
}

private void clearLinks() {
   link_model.clear();
}

private void appendGeneralLink(LispObject link, boolean reverse) {
   String link_desc = (String)link.getSlotValue(reverse ?
                                                "inverseDescriptor" : "descriptor");
   String obj_slot = reverse ? "from" : "to";
   Pcs_Identifier id = (Pcs_Identifier)link.getSlotValue(obj_slot);
   LispObject obj = (LispObject)archive_interface.getObject(id);
   String name = (String)obj.getSlotValue("printName");
   link_model.append("This is " + link_desc + " " + name + ".", id);
}

private void appendGeneralLink(LispObject link) {
   appendGeneralLink(link, false);
}

private boolean isAnnotationLink(LispObject link) {
   return (link instanceof Cds_AnnotationLink)
           || (link instanceof Cds_LispVersionAnnLink);
}

// Call this directly to refresh a node's children.
private AnnotationTreeNode nodeForAnnotationLink(AnnotationTreeNode node,
                                                 LispObject link) {
   node.removeAllChildren();
   Vector from_links = (Vector)link.getSlotValue("fromLinks");
   // Scan for child annotations.
   if (from_links != null) {
      for (int i = 0; i < from_links.size() - 1; i++) {
         Pcs_Identifier id = (Pcs_Identifier)from_links.elementAt(i);
         LispObject child_link = archive_interface.getObject(id);
         if (isAnnotationLink(child_link)) {
            node.add(nodeForAnnotationLink(child_link));
         }
      }
   }
   return node;
}
```

```
private AnnotationTreeNode nodeForAnnotationLink(LispObject link) {
   String summary = (String)link.getSlotValue("summary");
   AnnotationTreeNode node = new AnnotationTreeNode(summary, link);
   return nodeForAnnotationLink(node, link);
}

private void installAnnotationLinks(Vector links) {
   DefaultMutableTreeNode node = new DefaultMutableTreeNode();
   for (int i = 0; i < links.size(); i++) {
      node.add(nodeForAnnotationLink((LispObject)links.elementAt(i)));
   }
   tree.setModel(new DefaultTreeModel(node));
   annotation_text.setText("");
   selected_annotation = null;
   new_button.setEnabled(true);
   respond_button.setEnabled(false);
}

public void setLinks(LispObject resource) {
   this.resource = resource;
   annotation_text.setEditorKit(
      annotation_text.getEditorKitForContentType("text/html"));
   clearLinks();
   Vector all_links = new Vector();
   Vector annotation_links = new Vector();
   Vector from_link_ids = (Vector)resource.getSlotValue("fromLinks");
   LispObject[] from_links = null;
   LispObject[] to_links = null;
   if (from_link_ids != null) {
      from_link_ids = (Vector)from_link_ids.clone();
      // remove trailing null
      from_link_ids.removeElementAt(from_link_ids.size() - 1);
      from_links = archive_interface.getObjects(from_link_ids);
   }
   // Add inverse links if not already listed in forward direction.
   Vector to_link_ids = (Vector)resource.getSlotValue("toLinks");
   if (to_link_ids != null) {
      to_link_ids = (Vector)to_link_ids.clone();
       // remove trailing null
      to_link_ids.removeElementAt(to_link_ids.size() - 1);
      to_links = archive_interface.getObjects(to_link_ids);
   }

   // Performance hack, prefetch link targets
   Vector fetch = new Vector();
   if (from_links != null) {
      for (int i = 0; i < from_links.length; i++) {
         fetch.addElement(from_links[i].getSlotValue("to"));
      }
   }
   if (to_links != null) {
      for (int i = 0; i < to_links.length; i++) {
         fetch.addElement(to_links[i].getSlotValue("from"));
      }
   }
   archive_interface.getObjects(fetch);

   if (from_link_ids != null) {
      for (int i = 0; i < from_links.length; i++) {
         LispObject link = from_links[i];
         all_links.addElement(link);
         if (isAnnotationLink(link)) {
            annotation_links.addElement(link);
         }
         else {
            appendGeneralLink(link);
         }
      }
```

```
      }
      if (to_link_ids != null) {
         for (int i = 0; i < to_links.length; i++) {
            LispObject link = to_links[i];
            if (all_links.contains(link)) {
               continue;
            }
            else if (!isAnnotationLink(link)) {
               appendGeneralLink(link, true);
            }
         }
      }
     installAnnotationLinks(annotation_links);
   }

   public void actionPerformed(ActionEvent e) {
      JButton source = (JButton)e.getSource();
      if (source == new_button) {
         annotation_dialog.getAnnotation();
         if (annotation_dialog.annotation_type != AnnotationDialog.TYPE_CANCELLED) {
            archive_interface.annotateResource(resource, annotation_dialog);
            setLinks(archive_interface.refreshObject(resource));
         }
      }
      else if (source == respond_button) {
         annotation_dialog.getAnnotation();
         if (annotation_dialog.annotation_type != AnnotationDialog.TYPE_CANCELLED) {
            LispObject link =
               ((AnnotationTreeNode)tree.getLastSelectedPathComponent()).link;
            archive_interface.annotateResource(link, annotation_dialog);
            link = archive_interface.refreshObject(link);
            AnnotationTreeNode node =
               (AnnotationTreeNode)tree.getLastSelectedPathComponent();
            nodeForAnnotationLink(node, link);
            ((DefaultTreeModel)tree.getModel()).nodesWereInserted(node, new int[1]);
            ((DefaultTreeModel)tree.getModel()).nodeChanged(node);
         }
      }
   }

   public void valueChanged(ListSelectionEvent e) {
      if (e.getValueIsAdjusting())
         return;
      if (!link_list.isSelectionEmpty()) {
         navigator.displayResource(link_model.identifierAt(link_list.getSelectedIndex()));
      }
   }

   class AnnotationTreeNode extends DefaultMutableTreeNode {

      public LispObject link;

      AnnotationTreeNode(String s, LispObject link) {
         super(s);
         this.link = link;
      }

   }

}

}
```

11.7.9 code/java/Main.java

```
// Copyright 1998-99 Christopher R. Vincent (cvince@mit.edu)
```

```
// All Rights Reserved.

import java.awt.*;
import java.io.*;
import java.util.*;
import javax.swing.*;

public class Main extends JFrame {

public static int STANDARD_SPACING = 10;
public static int DIVIDER_SIZE = 3;
static String IMG_FILE = "code/java/date.jpg";
static String PROP_FILE = "configuration/java/config-data";

File properties_file;
Hashtable properties;

public Main() {
   super("DATE Client");
   properties = new Hashtable();
   properties_file = new File(PROP_FILE);
   loadProperties();
   JLabel picture = new JLabel();
   picture.setIcon(new ImageIcon(IMG_FILE));
   getContentPane().add(picture);
   pack();
   Dimension screen_size = Toolkit.getDefaultToolkit().getScreenSize();
   Dimension this_size = getSize();
   setLocation(screen_size.width - this_size.width, screen_size.height - this_size.height);
   setVisible(true);
   ConfigurationFrame config_frame = new ConfigurationFrame(this);
}

public synchronized void setProperty(String name, Object value) {
   properties.put(name, value);
}

public synchronized Object getProperty(String name) {
   return properties.get(name);
}

public synchronized void saveProperties() {
   try {
      FileOutputStream out = new FileOutputStream(properties_file);
      ObjectOutputStream p = new ObjectOutputStream(out);
      p.writeObject(properties);
      out.close();
   }
   catch(Exception e) {
      e.printStackTrace();
   }
}

public synchronized void loadProperties() {
   try {
      if (properties_file.canRead()) {
         FileInputStream in = new FileInputStream(properties_file);
         ObjectInputStream p = new ObjectInputStream(in);
         properties = (Hashtable)p.readObject();
         in.close();
      }
   }
   catch(Exception e) {
      e.printStackTrace();
   }
}

public static void main(String args[]) {
   try {
```

```
      UIManager.setLookAndFeel("com.sun.java.swing.plaf.mac.MacLookAndFeel");
   }
   catch (Exception e) {
      System.err.println("Error setting MacLookAndFeel.");
      e.printStackTrace();
   }
   Main app = new Main();
}

public static JPanel createRigidPadding(Component component, int size) {
   Dimension horiz_space = new Dimension(size, 0);
   Dimension vert_space = new Dimension(0, size);
   Box vert_padding = new Box(BoxLayout.Y_AXIS);
   vert_padding.add(Box.createRigidArea(vert_space));
   vert_padding.add(component);
   vert_padding.add(Box.createRigidArea(vert_space));
   JPanel outer_panel = new JPanel();
   outer_panel.setLayout(new BoxLayout(outer_panel, BoxLayout.X_AXIS));
   outer_panel.add(Box.createRigidArea(horiz_space));
   outer_panel.add(vert_padding);
   outer_panel.add(Box.createRigidArea(horiz_space));
   return outer_panel;
}

public static JPanel createRigidPadding(Component component) {
   return createRigidPadding(component, STANDARD_SPACING);
}

public static JPanel createBorderPanel(Component component, int padding) {
   Dimension horiz_space = new Dimension(padding, 0);
   Dimension vert_space = new Dimension(0, padding);
   Box vert_padding = new Box(BoxLayout.Y_AXIS);
   vert_padding.add(Box.createRigidArea(vert_space));
   vert_padding.add(component);
   vert_padding.add(Box.createRigidArea(vert_space));
   JPanel border_panel = new JPanel();
   border_panel.setLayout(new BoxLayout(border_panel, BoxLayout.X_AXIS));
   border_panel.add(Box.createRigidArea(horiz_space));
   border_panel.add(vert_padding);
   border_panel.add(Box.createRigidArea(horiz_space));
   border_panel.setBorder(BorderFactory.createEtchedBorder());
   return border_panel;
}

public static JPanel createBorderPanel(Component component) {
   return createBorderPanel(component, STANDARD_SPACING);
}

public static String capitalizeString(String s) {
   char c = s.charAt(0);
   if (Character.isUpperCase(c)) {
      return s;
   }
   else {
      return Character.toUpperCase(c) + s.substring(1);
   }
}

public static void beep() {
   Toolkit.getDefaultToolkit().beep();
}

}
```

11.7.10 code/java/Navigator.java

```
// Copyright 1998-99 Christopher R. Vincent (cvince@mit.edu)
// All Rights Reserved.

import java.awt.*;
import java.awt.event.*;
import java.net.URL;
import javax.swing.*;
import lisp.map.Pcs_Identifier;

public class Navigator extends JFrame {

public Navigator(ArchiveInterface archive_interface,
                 Pcs_Identifier archive_id,
                 ConfigurationFrame config) {
   super("Archive Navigator");
   Dimension screen_size = Toolkit.getDefaultToolkit().getScreenSize();
   setSize(new Dimension(screen_size.width,screen_size.height));
   /*
   addWindowListener(new WindowAdapter() {
      public void windowClosing(WindowEvent e) {System.exit(0);}
   });
   */
   NavigatorPanel panel = new NavigatorPanel(this, config, archive_interface, archive_id);
   getContentPane().add("Center", panel);
   panel.initMenuBar();
   setVisible(true);
}

}
```

11.7.11 code/java/NavigatorPanel.java

```
// Copyright 1998-99 Christopher R. Vincent (cvince@mit.edu)
// All Rights Reserved.

import java.awt.*;
import java.awt.event.*;
import java.net.*;
import java.util.*;
import javax.swing.*;
import lisp.lang.*;
import lisp.map.*;

public class NavigatorPanel extends JPanel {

HistoryFrame history_frame;
ArchiveInterface archive_interface;
CodeUnitPanel code_unit_panel;
LinkPanel link_panel;
Font small_font, monospace_font;
JMenuBar menu_bar;
public JFrame frame;
ConfigurationFrame configuration_frame;
ActionListener menu_listener;
public Color subtle_select_color;
Pcs_Identifier archive_id;
Hashtable collection_table;
Vector collection_vect;

public NavigatorPanel(JFrame frame,
                      ConfigurationFrame config,
                      ArchiveInterface archive_interface,
                      Pcs_Identifier archive_id) {
   this.frame = frame;
```

```
   configuration_frame = config;
   this.archive_interface = archive_interface;
   this.archive_id = archive_id;
   subtle_select_color = new Color(200,200,255);
   menu_listener = new NavigatorMenuListener();
   small_font = new Font("Geneva", Font.PLAIN, 9);
   monospace_font = new Font("Courier", Font.PLAIN, 12);
   link_panel = new LinkPanel(this, archive_interface);
   code_unit_panel = new CodeUnitPanel(this, link_panel, archive_interface, archive_id);
   history_frame = new HistoryFrame(this);
   setLayout(new BoxLayout(this, BoxLayout.Y_AXIS));
   JSplitPane split_pane = new JSplitPane(JSplitPane.HORIZONTAL_SPLIT,
                                          code_unit_panel,
                                          link_panel);
   add(split_pane);
   // percentage version of setDividerLocation doesn't work before the pane is sized...
   split_pane.setDividerLocation((int)Math.round(0.66 * frame.getSize().width));
   // Fetch collections for this archive.
   collection_table = new Hashtable();
   collection_vect = archive_interface.getArchiveCollections(archive_id);
   collection_vect.removeElementAt(collection_vect.size() - 1);  // trailing null
   for (int i = 0; i < collection_vect.size(); i++) {
      Pcs_Identifier id = (Pcs_Identifier)collection_vect.elementAt(i);
      LispObject collection = archive_interface.getObject(id);
      collection_table.put((String)collection.getSlotValue("title"), collection);
   }
}

public void displayResource(Pcs_Identifier id) {
   LispObject obj = archive_interface.getObject(id);
   history_frame.appendHistory(obj);
   code_unit_panel.displayResource(obj);
}

public void initMenuBar() {
   menu_bar = new JMenuBar();
   JMenu menu = new JMenu("Navigator");
   menu.setMnemonic('N');
   JMenuItem item = new JMenuItem("New", 'N');
   item.addActionListener(menu_listener);
   menu.add(item);
   item = new JMenuItem("Quit", 'Q');
   item.addActionListener(menu_listener);
   item.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_Q, ActionEvent.CTRL_MASK));
   menu.add(item);
   menu_bar.add(menu);
   menu = new JMenu("Collections");
   menu.setMnemonic('C');
   for (Enumeration enum = collection_vect.elements(); enum.hasMoreElements();) {
      LispObject collection =
         archive_interface.getObject((Pcs_Identifier)enum.nextElement());
      item = new JMenuItem((String)collection.getSlotValue("title"));
      item.addActionListener(menu_listener);
      menu.add(item);
   }
   menu_bar.add(menu);
   menu = new JMenu("Windows");
   menu.setMnemonic('W');
   item = new JMenuItem("History", 'H');
   item.addActionListener(menu_listener);
   item.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_H, ActionEvent.CTRL_MASK));
   menu.add(item);
   menu_bar.add(menu);
   frame.setJMenuBar(menu_bar);
}

class NavigatorMenuListener implements ActionListener {

   public void actionPerformed(ActionEvent e) {
```

```
      JMenuItem source = (JMenuItem)(e.getSource());
      String s = source.getText();
      if (s.equals("History")) {
         history_frame.setVisible(true);
         history_frame.toFront();
      }
      else if (s.equals("New")) {
         configuration_frame.setVisible(true);
         configuration_frame.toFront();
      }
      else if (s.equals("Quit")) {
         System.exit(0);
      }
      else {
         LispObject collection = (LispObject)collection_table.get(s);
         if (collection != null) {
            code_unit_panel.displayCollection(collection);
         }
      }
   }

}

}
```

11.7.12 code/java/lisp/lang/Keyword.java

```
// Copyright 1998-99 Christopher R. Vincent (cvince@mit.edu)
// All Rights Reserved.

package lisp.lang;

import java.io.Serializable;

public class Keyword implements Serializable {

private static final long serialVersionUID = 1;

public String name;

public Keyword(String name1) {
   name = name1;
}

public Keyword(Keyword keyword) {
   name = keyword.name;
}

public boolean equals(Keyword obj) {
   return name.equals(obj.name);
}

public int hashCode(Keyword obj) {
   return obj.name.hashCode();
}

public String toString() {
   return ":"+name;
}

}
```

11.7.13 code/java/lisp/lang/LispObject.java

```
// Copyright 1998-99 Christopher R. Vincent (cvince@mit.edu)
// All Rights Reserved.

package lisp.lang;

public class LispObject {

public Object getSlotValue(String name) {
   try {
      return getClass().getField(name).get(this);
   }
   catch (Exception e) {
      System.err.println("Could not access field named "+name+" for "+this);
      e.printStackTrace();
      return null;
   }
}

}
```

11.7.14 code/java/lisp/lang/Symbol.java

```
// Copyright 1998-99 Christopher R. Vincent (cvince@mit.edu)
// All Rights Reserved.

package lisp.lang;

import java.io.Serializable;

public class Symbol implements Serializable {

private static final long serialVersionUID = 1;

public String pkg;
public String name;

public Symbol(String pkg1, String name1) {
   pkg = pkg1;
   name = name1;
}

public Symbol(Symbol symbol) {
   pkg = symbol.pkg;
   name = symbol.name;
}

public String toString() {
   return pkg + ":" + name;
}

}
```

11.8 Demonstration Archive

11.8.1 demo/archive-cl-http.lisp

```
;;; -*- Mode: lisp; Syntax: ANSI-Common-Lisp; Base: 10; Package: CDS -*-
;;;
;;; Copyright 1998-99 Christopher R. Vincent (cvince@mit.edu)
```

```
;;; All Rights Reserved.
;;;
;;;---------------------------------------------------------------------
;;;
;;; ARCHIVE CL-HTTP
;;;

(in-package :cds)

(defparameter *cl-http-portable-sources*
  '(("http:server;preliminary")
    ("http:server;variables")
    ("http:server;package")
    ("http:server;base64-encoding")
    ("http:server;md5")
    ("http:server;sha")
    ("http:smtp;package")
    ("http:smtp;smtp")
    ("http:smtp;mail")
    ("http:server;task-queue")
    ("http:server;class")
    ("http:server;url-class")
    ("http:server;plist")
    ("http:server;utils")
    ("http:server;tokenizer")
    ("http:server;headers")
    ("http:server;host")
    ("http:server;url")
    ("http:server;html2")
    ("http:server;netscape-1-1")
    ("http:server;vrml-1-0")
    ("http:server;image-maps")
    ("http:server;netscape-2-0")
    ("http:server;netscape-3-0")
    ("http:server;html-3-2")
    ("http:server;scripts")
    ("http:server;netscape-4-0")
    ("http:server;shtml")
    ("http:server;http-conditions")
    ("http:server;log")
    ("http:server;authentication")
    ("http:server;data-cache")
    ("http:server;server")
    ("http:server;cgi")
    ("http:server;preferences")
    ("http:server;web-configuration")
    ("http:clim;package")
    ("http:clim;interface")
    ; client system
    ("http:client;sexp-browser")
    ; html-parser system
    ("http:html-parser;v9;packages")
    ("http:html-parser;v9;defs")
    ("http:html-parser;v9;patmatch")
    ("http:html-parser;v9;rewrite-engine")
    ("http:html-parser;v9;rewrite-rules")
    ("http:html-parser;v9;html-tags")
    ("http:html-parser;v9;html-reader")
    ("http:html-parser;v9;html-parser")
    ("http:html-parser;v9;html-utilities")
    ; lambda-ir system
    ("http:lambda-ir;package")
    ("http:lambda-ir;ir-utils")
    ("http:lambda-ir;variables")
    ("http:lambda-ir;class")
    ("http:lambda-ir;bit-vectors")
    ("http:lambda-ir;data-structures")
    ("http:lambda-ir;computations")
    ("http:lambda-ir;ir-base")
```

```
    ("http:lambda-ir;contexts")
    ("http:lambda-ir;constraints")
    ("http:lambda-ir;bin-dump-utils")
    ; proxy system
    ("http:proxy;utils")
    ("http:proxy;database")
    ("http:proxy;cache")
    ("http:proxy;proxy-cache")
    ("http:proxy;proxy")
    ("http:proxy;documentation")
    ; w3p system
    ("http:w3p;package")
    ("http:w3p;class")
    ("http:w3p;w3p-system")
    ("http:w3p;functions")
    ("http:w3p;standard-types")
    ("http:w3p;standard-methods")
    ("http:w3p;html-definitions")
    ; w4 system
    ("http:client;w4-client")
    ("http:w4;package")
    ("http:w4;variables")
    ("http:w4;utils")
    ("http:w4;class")
    ("http:w4;walker")
    ("http:w4;constraints")
    ("http:w4;actions")
    ("http:w4;activity")))

#|| There are still problems with duplicate definitions in ACL.
(defparameter *acl-cl-http-sources*
  '(("http:acl;start")
    ("http:acl;startpc")
    ("http:acl;transhosts")
    ("http:acl;translations")
    ("http:acl;aclpc;sysdcl" . aclpc)
    ("http:acl;aclpc;format" . aclpc)
    ("http:acl;aclpc;stream-sequence" . aclpc)
    ("http:acl;aclpc;listener" . aclpc)
    ("http:acl;aclpc;mintface" . aclpc)
    ("http:acl;server;sysdcl")
    ("http:acl;aclpc;ipc" . aclpc)
    ("http:acl;acl5;ipc" . acl5)
    ("http:acl;server;multivalent-stream" . (not acl5))
    ("http:acl;server;tcp-stream-svr3" . (and Allegro-version>= (version< 4 3)))
    ("http:acl;server;tcp-stream" . (and Allegro-version>= (version>= 4 3) (not ACL5)))
    ("http:acl;server;stream-sequence-svr3" . (and Allegro-version>= (version< 4 3)))
    ("http:acl;server;stream-sequence" . (and Allegro-version>= (version>= 4 3) (not ACL5)\
))
    ("http:acl;server;stream-chunk" . (not ACL5))
    ("http:acl;server;chunk-transfer" . (not acl5))
    ("http:acl;server;unix" . (or unix aclpc acl5))
    ("http:acl;server;url")
    ("http:acl;server;update")
    ("http:acl;server;cgi")
    ("http:acl;server;proxy")
    ("http:acl;server;tcp-interface")
    ("http:acl;server;patched")
    ("http:acl;aclpc;patched" . aclpc)
    ("http:acl;acl5;patched" . acl5)
    ; client system
    ("http:acl;client;sysdcl")
    ("http:acl;client;unix")))
||#

(defparameter *mcl-cl-http-sources*
  '(("http:mcl;start-server")
    ("http:mcl;start-server-appletalk")
    ("http:mcl;start-server-ip-number")
```

```
    ("http:mcl;mcl-configuration")
    ("http:mcl;translations")
    ("http:mcl;envdcl")
    ("http:mcl;server;sysdcl")
    ("http:mcl;server;store-conditional" . (and ccl-3 (not (or ppc-target ccl-3.1))))
    ("http:mcl;server;process-poll" . (and ccl-3 (not process-poll)))
    ("http:mcl;server;lock-multiple-readers" . (and ccl-3 (not :multiple-reader-locks)))
    ("http:mcl;server;task-process" . ccl-3)
    ("http:mcl;server;xmactcp" . (and ccl-3 (not Open-Transport)))
    ("http:mcl;server;xio-buffer" . (and Open-Transport (not (or CCL-4.2 ccl-3.3))))
    ("http:mcl;server;xopentransport" . (and Open-Transport (not (or CCL-4.2 ccl-3.3))))
    ("http:mcl;server;mcl-clos-patch" . (and ppc-target (not ccl-4)))
    ("http:mcl;server;timers" . (or ccl-3 ccl-3.1 ccl-4))
    ("http:mcl;server;resources")
    ("http:mcl;server;tcp-stream" . (not Open-Transport))
    ("http:mcl;server;tcp-ot-stream" . Open-Transport)
    ("http:mcl;server;tcp-conditions2" . (not ccl-3))
    ("http:mcl;server;www-utils")
    ("http:mcl;server;mcl")
    ("http:mcl;server;log-window")
    ("http:mcl;server;tcp-interface2" . (not ccl-3))
    ("http:mcl;server;tcp-interface" . ccl-3)
    ("http:mcl;server;control")
    ("http:mcl;server;cl-http-menu")
    ; client system
    ("http:mcl;client;sysdcl-substrate")
    ("http:mcl;client;sysdcl")
    ; html-parser system
    ("http:mcl;html-parser;sysdcl")
    ; lambda-ir system
    ("http:mcl;lambda-ir;sysdcl")
    ; proxy system
    ("http:mcl;proxy;sysdcl")
    ("http:mcl;proxy;patches")
    ("http:mcl;proxy;mcl")
    ; w3p system
    ("http:mcl;w3p;sysdcl")
    ; w4 system
    ("http:mcl;w4;sysdcl")))

(defparameter *genera-cl-http-sources*
  '(("http:lispm;server;sysdcl")
    ("http:lispm;server;tcp-lispm-stream")
    ("http:lispm;server;tcp-patch-hang-on-close")
    ("http:lispm;server;lispm")
    ; client system
    ("http:lispm;client;sysdcl")
    ("http:lispm;client;sysdcl-substrate")
    ("http:lispm;client;images-lispm")
    ("http:lispm;client;client")
    ("http:lispm;client;lispm")
    ; lambda-ir system
    ("http:lispm;lambda-ir;sysdcl")
    ; proxy system
    ("http:lispm;proxy;sysdcl")
    ; w4 system
    ("http:lispm;w4;sysdcl")
    ("http:lispm;w4;receive-tcp-segment-patch")))

(defparameter *cl-http-platform-specific-sources*
  '(
    ; (:acl . *acl-cl-http-sources*)))
    (:mcl . *mcl-cl-http-sources*)
    (:genera . *genera-cl-http-sources*)))

(defun version-for-code-unit (code-unit implementation-name version-number)
  (let ((implementation (cond (implementation-name (get-implementation  implementation-nam\
e code-unit))
                              ; Choose at random.
```

```
                              (t (loop for name being the hash-key in (code-unit-implement\
ations code-unit)
                                       do (return (gethash name (code-unit-implementations\
 code-unit))))))))
    (cond (version-number (get-resource-version implementation version-number))
          (t (get-newest-resource-version implementation)))))

(defun find-code-unit-version (archive package code-unit-name type implementation-name ver\
sion-number)
  (let ((code-unit (car (find-indexed-resource archive (cons package code-unit-name) :type\
 type))))
    (version-for-code-unit code-unit implementation-name version-number)))

(defun create-example-annotations (archive)
  (let* ((version1 (find-code-unit-version archive :www-utils "ip-address-for-host-domain-\
name" :function "mcl" nil))
         (source-code (copy-seq (version-source-code version1)))
         (parse-host-start (search "%parse-host-address" source-code))
         (parse-host-end (+ parse-host-start (length "%parse-host-address")))
         (implementation (version-implementation version1))
         (version3 (progn
                     ; Change %parse-host-address to parse-host.
                     (make-lisp-source-code-version implementation
                                                    (concatenate 'string (subseq source-co\
de 0 parse-host-start)
                                                                 "parse-host"
                                                                 (subseq source-code parse\
-host-end)))
                     ; Change it back.
                     (make-lisp-source-code-version implementation (version-source-code ve\
rsion1))))
         (annotation (first (annotate-version version3
                                              "This should call <b>parse-host</b> instead \
of
referencing <b>%parse-host-address</b> directly."
                                              "aph@ai.mit.edu"
                                              :summary "Wrong abstraction used?")))
         (response1 (first (annotate-resource annotation
                                              "No, performance is <font color=red>critical\
</font> here.
Calling <b>parse-host</b> was too slow."
                                              "bb@ai.mit.edu"
                                              :summary "No, performance trade-off.")))
    (annotate-resource response1
                       "That was with the old hardware, network latency now dominates perf\
ormance."
                       "aph@ai.mit.edu"
                       :summary "Requirements have changed.")))

(defun define-new-function (archive package name-string source-code)
  (let* ((code-unit (make-lisp-code-unit archive package
                                         :class 'lisp-function
                                         :name (cons package name-string)))
         (implementation (make-lisp-implementation code-unit "Default"
                                                   :save-implementation-p nil
                                                   :save-code-unit-p nil))
         (version (make-lisp-source-code-version implementation source-code
                                                 :save-implementation-p nil
                                                 :save-version-p nil)))
    (save-objects (list code-unit implementation version))
    (register-resource code-unit archive)
    code-unit))

(defun crlf-encode-string (string)
  (loop for c across string
        collect c into result
        when (eq c #\Return)
        collect #\Linefeed into result
        finally (return (coerce result 'string))))
```

```
(defun create-example-test-link (archive)
  (let* ((test-subject (car (find-indexed-resource archive '(:www-utils . "ip-address-for-\
host-domain-name") :type :function)))
         (test-code-unit (define-new-function archive :www-utils "test-ip-address-for-host\
-domain-name"
                           (crlf-encode-string "(defun test-ip-address-for-host-domain-nam\
e ()
  \"Returns non-nil if the test is successful.\"
  ; This end-to-end test depends on particular DNS bindings.
  (loop for (host-name . ip-address) in (compute-domain-name-resolution-test-set)
        unless (string-equal ip-address (ip-address-for-host-domain-name host-name))
        do (return nil)
        finally (return t)))"))))
    (link-objects 'test-case-link test-subject test-code-unit)
    (define-new-function archive :www-utils "compute-domain-name-resolution-test-set"
      (crlf-encode-string "(defun compute-domain-name-resolution-test-set ()
  \"Return a list of host-name-string, ip-address-string pairs.\"
  ; This could dynamically select a random test set from a DNS database.
  '((\"ai.mit.edu\" . \"128.52.32.80\")
    (\"mit.edu\" . \"18.72.0.100\")))"))))


(defun create-example-collections (archive)
  (flet ((spec-to-versions (specs)
           (mapcar #'(lambda (args) (apply 'find-code-unit-version archive args)) specs)))
    (let ((dns-support (spec-to-versions '((:www-utils "parse-host" :function "mcl" nil)
                                           (:www-utils "parse-internet-addresses" :functio\
n "mcl" nil)
                                           (:www-utils "ip-address-for-host-domain-name" :\
function "mcl" nil)
                                           (:www-utils "%parse-host-address" :function "mc\
l" nil))))
          (object-printers (mapcar #'(lambda (code-unit) (funcall 'version-for-code-unit c\
ode-unit nil nil))
                                   (find-indexed-resource archive '(:http . "print-object"\
))))
          (urls (spec-to-versions '((:url "url" :class nil nil)
                                    (:url  "http-url" :class nil nil)
                                    (:url "local-url-p" :generic-function nil nil)
                                    (:http "export-url" :generic-function nil nil)
                                    (:http "show-url" :generic-function nil nil)))))
      (make-lisp-code-collection archive "HTTP Object Writers" object-printers)
      (make-lisp-code-collection archive "Uniform Resource Locators" urls)
      (make-lisp-code-collection archive "Domain Name Service Support" dns-support))))

(defun archive-cl-http-sources (&key archive (optimize-storage t)
                                     &aux (count 0))
  (or archive
      (setf archive (get-archive :cl-http))
      (setf archive (make-lisp-code-archive :cl-http "Common Lisp Hypermedia Server")))
  (format t "~%; Parsing CL-HTTP sources.")
  (labels ((coerce-to-keyword-list (thing)
             (etypecase thing
               (cons (mapcar #'coerce-to-keyword-list thing))
               (symbol (intern-keyword (symbol-name thing)))
               (integer thing)))   ; acl allows numbers
           (process-sources (sources implied-conditional)
             (loop for (name . conditional) in sources
                   for pathname = (pathname (concatenate 'string name ".lisp"))
                   do (when conditional (setq conditional (coerce-to-keyword-list conditio\
nal)))
                   (cond ((and implied-conditional conditional)
                          (setq conditional (list :and implied-conditional conditional)))
                         (implied-conditional
                          (setq conditional implied-conditional)))
                   (tagbody
                     parse-file
                     (restart-case
```

```
                    (incf count (parse-lisp-file pathname archive
                                                 :implied-conditional conditional
                                                 :save-archive-p nil))
                    (retry-file () :report (lambda (s) (format s "Retry parsing file ~S\
." pathname))
                                (go parse-file))
                    (skip-file () :report (lambda (s) (format s "Skip parsing file ~S."\
 pathname))))))))
    (declare (inline walk-symbol-list coerce-to-keyword-list process-sources))
    (process-sources *cl-http-portable-sources* nil)
    (loop for (conditional . sources) in *cl-http-platform-specific-sources*
          do (process-sources (symbol-value sources) conditional))
    (save-object archive))
  (format t "~%; Parsed ~D Lisp forms." count)
  (format t "~%; Installing implicit code unit links...")
  (ensure-implicit-links archive)
  (format t "~%; Creating example test cases...")
  (create-example-test-link archive)
  (format t "~%; Creating example annotations...")
  (create-example-annotations archive)
  (format t "~%; Creating example collections...")
  (create-example-collections archive)
  (when optimize-storage
    (format t "~%; Optimizing persistent storage...")
    (optimize-storage-mechanism :verbose t)))

(defun create-example-collections (archive)
  (flet ((spec-to-versions (specs)
           (mapcar #'(lambda (args) (apply 'find-code-unit-version archive args)) specs)))
    (let ((dns-support (spec-to-versions '((:www-utils "parse-host" :function "mcl" nil)
                                           (:www-utils "parse-internet-addresses" :functio\
n "mcl" nil)
                                           (:www-utils "ip-address-for-host-domain-name" :\
function "mcl" nil)
                                           (:www-utils "%parse-host-address" :function "mc\
l" nil))))
          (object-printers (mapcar #'(lambda (code-unit) (funcall 'version-for-code-unit c\
ode-unit nil nil))
                                   (find-indexed-resource archive '(:http . "print-object"\
))))
          (urls (spec-to-versions '((:url "url" :class nil nil)
                                    (:url  "http-url" :class nil nil)
                                    (:url "local-url-p" :generic-function nil nil)
                                    (:http "export-url" :generic-function nil nil)
                                    (:http "show-url" :generic-function nil nil)))))
      (make-lisp-code-collection archive "HTTP Object Writers" object-printers)
      (make-lisp-code-collection archive "Uniform Resource Locators" urls)
      (make-lisp-code-collection archive "Domain Name Service Support" dns-support))))

(defun archive-cl-http-sources (&key archive (optimize-storage t)
                                     &aux (count 0))
  (or archive
      (setf archive (get-archive :cl-http))
      (setf archive (make-lisp-code-archive :cl-http "Common Lisp Hypermedia Server")))
  (format t "~%; Parsing CL-HTTP sources.")
  (labels ((coerce-to-keyword-list (thing)
             (etypecase thing
               (cons (mapcar #'coerce-to-keyword-list thing))
               (symbol (intern-keyword (symbol-name thing)))
               (integer thing)))   ; acl allows numbers
           (process-sources (sources implied-conditional)
             (loop for (name . conditional) in sources
                   for pathname = (pathname (concatenate 'string name ".lisp"))
                   do (when conditional (setq conditional (coerce-to-keyword-list conditio\
nal)))
                   (cond ((and implied-conditional conditional)
                          (setq conditional (list :and implied-conditional conditional)))
                         (implied-conditional
                          (setq conditional implied-conditional)))
```

```
                  (tagbody
                    parse-file
                    (restart-case
                      (incf count (parse-lisp-file pathname archive
                                                   :implied-conditional conditional
                                                   :save-archive-p nil))
                      (retry-file () :report (lambda (s) (format s "Retry parsing file ~S\
." pathname))
                                  (go parse-file))
                      (skip-file () :report (lambda (s) (format s "Skip parsing file ~S."\
 pathname))))))))
    (declare (inline walk-symbol-list coerce-to-keyword-list process-sources))
    (process-sources *cl-http-portable-sources* nil)
    (loop for (conditional . sources) in *cl-http-platform-specific-sources*
          do (process-sources (symbol-value sources) conditional))
    (save-object archive))
  (format t "~%; Parsed ~D Lisp forms." count)
  (format t "~%; Installing implicit code unit links...")
  (ensure-implicit-links archive)
  (format t "~%; Creating example test cases...")
  (create-example-test-link archive)
  (format t "~%; Creating example annotations...")
  (create-example-annotations archive)
  (format t "~%; Creating example collections...")
  (create-example-collections archive)
  (when optimize-storage
    (format t "~%; Optimizing persistent storage...")
    (optimize-storage-mechanism :verbose t)))

(defun count-cl-http-code-lines (&aux (line-count 0) (version-count 0))
  "Count total lines of source code."
  (labels ((count-version (version)
             (incf version-count)
             (typecase version
               (lisp-code-container-version)
               (t (let ((code (version-source-code version))
                        (start 0))
                    (loop while start
                          do (incf line-count)
                          (setq start (search '(#\Return #\Linefeed) code :start2 start))
                          (when start (incf start)))))))
           (count-implementation (implementation)
             (map-resource-versions #'count-version implementation))
           (count-code-unit (name id)
             (declare (ignore name))
             (map-implementations #'count-implementation (load-object id))))
    (map-indexed-resources #'count-code-unit (get-archive :cl-http))
    (format t "~%; Counted ~D versions, ~D total lines of code." version-count line-count)\
))

(defun count-cl-http-multiple-implementations (&aux (count 0))
  "Count number of code units which have more than one implementation."
  (flet ((count-code-unit (name id)
             (declare (ignore name))
             (when (> (hash-table-count (code-unit-implementations (load-object id))) 1)
               (incf count))))
    (map-indexed-resources #'count-code-unit (get-archive :cl-http))
    (format t "~%; Counted ~D code units with multiple implementations." count)))
```

11.8.2 demo/build-server-image.lisp

```
;;; -*- Mode: lisp; Syntax: ANSI-Common-Lisp; Base: 10; Package: CL-USER -*-
;;;
;;; Copyright 1998-99 Christopher R. Vincent (cvince@mit.edu)
;;; All Rights Reserved.
;;;
```

```
;;;-------------------------------------------------------------------
;;;
;;; BUILD AN IMAGE
;;;

(in-package :cl-user)

; Load this file in a clean Lisp, after any environment customizations you
; want preserved in the image.  Then place the "start-server" image in your
; MCL directory.

; See archive-cl-http.lisp for building an example database once
; the server is running.

(unless *load-pathname*
  (error "File must be loaded such that *load-pathname* is non-NIL."))

#-MCL
(error "Port this file to your platform.")

#+MCL
(let* ((mcl-dir (translate-logical-pathname *load-pathname*))
       (root-path (directory-namestring (make-pathname :device (pathname-device *load-path\
name*)
                :directory (butlast (pathname-directory *load-pathname*) 1)))))

  ; Load CL-HTTP without any configuration info.
  (load (merge-pathnames ";cl-http-67-106;mcl;envdcl" root-path) :verbose t)

  ; Load the DATE server, without initializing persistent storage.
  (load (merge-pathnames ";configuration;platform;mcl;sysdcl" root-path) :verbose t)

  ; Save an application image.  The init file configures CL-HTTP and
  ; performs exports needed by the DATE server.
  (save-application (merge-pathnames ";start-server" root-path)
                    :init-file (merge-pathnames ";configuration;platform;mcl;cl-http;cl-ht\
tp-init" root-path)
                    :size 33554432))
```