Naming and sharing resources across administrative boundaries

Volume Two
Software documentation and experimental data

A Dissertation

Submitted to the Faculty

in partial fulfillment of the requirements for the

degree of

Doctor of Philosophy

in

Computer Science

by

Jonathan R. Howell

DARTMOUTH COLLEGE

Hanover, New Hampshire

26 May 2000

Examining Committee:

David Kotz (chairman)

Robert Gray

Doug McIlroy

Margo Seltzer

Roger Sloboda
Dean of Graduate Studies

Contents

Appendix E

Snowflake software documentation

This appendix documents the software described in the body of the dissertation; it includes packages that implement and packages that use the Snowflake infrastructure.

This overview illustrates the use of the Snowflake components by highlighting some example application code and indicating the Snowflake interfaces the code uses.

Some package and class documentation are omitted to save space. Often this is done because they are deprecated, such as early prototypes. Some classes have shortened descriptions because their method list is simply the set of methods required to implement a superclass or interface.

Naming.

The first illustration consists of excerpts from `ide.Shell`, the command-line user interface to Snowflake-named resources, and gives examples of both name lookup and name binding. First, the Shell sets up its own namespace from first principles.

```
public static void main(String args[]) {
    ROSOut rosout = new ROSOut(System.out);
    RISIn risin = new RISIn(System.in);
        // Create remotely-accessible versions of the terminal
        // I/O streams.
    Namespace root = new LocalNS();
        // Create the root namespace for this shell; Programs
        // we invoke will basically share it.
    Namespace cmd = new LocalNS();
    root.bind("cmd", cmd);
        // Create a /cmd directory to hold bootstrap commands,
        // and bind it into the root.
    cmd.bind("mkrem", new mkrem());
    cmd.bind("ls", new ls());
        // Bootstrap commands include mkrem, for binding new, raw
        // (low-level) resources, and ls, for exploring.
    Namespace streams = new LocalNS();
```

```
        root.bind("streams", streams);
        streams.bind("stdin", risin);
        streams.bind("stdout", rosout);
            // Bind the I/O streams into the root namespace.
        Sf.pushNamespace(root);
            // Install the root namespace as the default namespace
            // for this thread; any invocations of the sf.Sf
            // tools will refer to it implicitly.
        shell();
            // Run the main loop.
    }
```

A fancier shell might read an `.rc` file before proceeding to accept user input. Notice that there is no $PATH variable; the single name `/cmd` resolves all command lookups.

The main loop of the shell simply parses users commands, locates the desired `sf.Program` resource, and invokes it with conventional resources configured into its namespace. Hence Programs are invoked with only a single parameter, the Program's root namespace.

```
    /**
     * The main Program loop of the shell. Retrieves its I/O streams
     * from the Snowflake namespace, and loops processing commands.
     */
    public static void shell() {
        RemoteOutputStream ros =
            (RemoteOutputStream) Sf.lookupPath("streams/stdout");
        RemoteInputStream ris =
            (RemoteInputStream) Sf.lookupPath("streams/stdin");
        ...
            // Retrieve the I/O streams from the namespace (sensible
            // when called other than from the Unix command line), and
            // wrap them for easy use with java.io classes.
        while (!done) {
            pw.print("% ");      // prompt
            ...                  // input command and parse into words
            if (!verb.startsWith("/")) {
                // rooted command names are relative to the namespace root;
                // others name commands in the /cmd directory.
                verb="/cmd/"+verb;
            }
            Object ob = Sf.lookupPath(verb);
                // Look up the command in the root namespace.
            Namespace ns = (Namespace) Sf.lookupPath("/");
            Namespace arg = ns;
            arg.bind("argv", cargv);
```

```
            // Bind the arguments to the name "argv" in the Program's
            // namespace
        ((Program) ob).run(arg);
            // Now that the Program's root contains all of its
            // arguments, simply invoke the Program, passing it its new
            // root namespace (context).
    }
}
```

Sharing and Security.

The following examples explore the interface to Snowflake's logical model for sharing and securing resources.

An RMI server application. This snippet appears in `relational.SSHDatabase` to configure a relational database object that demands Snowflake-style authorization:

```
SSHContext context = SSHContext.getDefault();
    // Get the default SSHContext object; we will use it
    // to handle incoming requests.
Database theDatabase =
    new InternalDatabase(context, serverPublicKey);
    // Create a new relational database server.
    // Instruct it to accept its requests only over SSH using the
    // given context.
    // The serverPublicKey argument is a SDSIPrincipal (parsed from the
    // command line) that identifies the issuer. Any client must show
    // its authority over the issuer.
InetAddress thisHost = InetAddress.getLocalHost();
Naming.rebind("//"+thisHost.getHostName()
    +"/RMIDatabase", theDatabase);
    // Bind the resource into a public name space. In this example,
    // I publicize the database with the Java RMI Registry for
    // expediency.
```

The corresponding code in `relational.InternalDatabase` implements the authorization requirements:

```
public InternalDatabase(SSHContext context,
    SDSIPrincipal serverIssuer)
    throws RemoteException {
    super(0 /*port*/,
        new SSHClientSocketFactory(context),
        new SSHServerSocketFactory(context));
            // Tell the superclass constructor to only
            // accept RMI connections over SSH channels,
```

```
                // using the keys defined in the given context.
        initAuthorization(serverIssuer);
                // Store the required issuer for this service.
    }
    void initAuthorization(SDSIPrincipal serverIssuer) {
        SexpList databaseTag =
            (new SexpList()).append("database").append("mine");
        SexpList sl =
            (new SexpList()).append("tag").append(databaseTag);
        this.requestTag = new Tag(sl);
            // Construct a prototype tag that describes the minimum
            // authority required of any client.
        this.serverIssuer = serverIssuer;
            // Take note of the required issuer.
    }
```

Each of the implementations of Remote methods call checkAuth() before honoring their requests. This insertion of checks is trivial and could easily be made mechanical. Better yet, it could be done in the RMI remote reference layer with sufficient plumbing.

```
    public void insert(Relational[] ros) {
        checkAuth();
        ...
    }
    public void createIndex(FieldDescriptor fd) {
        checkAuth();
        ...
    }
    public ResultSet evaluateSelect(Select s) {
        checkAuth();
        ...
    }
```

The `checkAuth()` method tests that the client has the required authority.

```
    void checkAuth() {
        ssh.RSA.RSAKey k = SSHSocket.whoCalledMe();
        SDSIRSAPublicKey subject = new SDSIRSAPublicKey(k);
            // Determine the "subject" — the principal that
            // actually made the request.
        try {
            Proof proof
                = OneLineCacheRecipient.getCachedProof(subject);
                // See if the subject's proof has already been delivered.
            if (proof==null) {
                throw new InvalidProofException("no proof found");
```

```
                // If not, demand proof from the client.
            }
            proof.verify(serverIssuer, subject, requestTag);
                // Verify the proof we have on hand, and verify that it
                // indeed supports the request (using the prototype tag
                // created earlier). After the first verification, this
                // operation becomes very fast.
            return;
        } catch (InvalidProofException ex) {
            throw new SfNeedAuthorizationException(serverIssuer,
                subject, requestTag,
                OneLineCacheRecipient.getRecipient(), ex.toString());
                // Convert any error into a demand for a proof
                // of authority from the client.
        }
    }
```

That's all there is to a simple server.

An RMI client application. Here is the corresponding access code in a simple client:

```
public static void main(String argv[]) {
    SSHContext myContext = SSHContext.newKeys();
        // Generate a fresh key pair for communication
    Prover2 prover = new Prover2("certs");
        // Initialize the prover tool using a stash of delegations
        // known to the client

    SDSIKeyPair skp = new SDSIKeyPair(myContext.getPrivateKey(),
        myContext.getPublicKey());
    prover.introduceObject(skp);
        // introduce the SSH channel keys as a principal to the
        // Prover, so that it can write delegations to the SSH channel
        // when necessary.
    prover.loadCache();
        // Have the prover bootstrap its delegations from the stash
        // indicated in its constructor.
    InvokeHack.setCurrentProver(prover);
        // Install the prover as the active callback that handles
        // demands of authority for RMI requests.
    ...
}
```

Notice that the client only need initialize its Prover and channel mechanism. No other client code is changed. The client application accesses RMI objects just like any other, and in the course of accessing objects whose references involve a Snowflake-protected server object, the Prover automatically constructs the appropriate proofs of the client's authority.

An HTTP-to-RMI gateway application. The final example illustrates the Snowflake calls made by the email gateway illustrated in Section 11.3. Since the gateway is a client of the RMI database server, it begins with setup code very similar to that of the previous example.

The gateway code itself, `servlet.MailServlet`, inherits from `servlet.ProtectedServlet`, the class that implements the basic server-side functionality of the Snowflake HTTP signed-requests protocol described in 10.3.3.

On each access, the gateway associates a specific SSH context with its current thread, to ensure that only requests made by the current thread (in service of a specific client) acquire the authority delegated to that channel, and that when the gateway has finished servicing this client's request, the authority used is revoked. This code represents a scoped construct for amplification of rights.

```
public void doGet(HttpServletRequest request,
    HttpServletResponse response)
    throws ServletException, IOException {
    try {
        SSHContext.contextByThread.set(myContext);
            // set up outgoing SSH context for this thread.
        (new Handler(request,response)).doGet();
            // Handle the request
    } finally {
        SSHContext.contextByThread.set(null);
            // Don't let other users of this thread borrow my context.
    }
}
```

The `getRequiredIssuer()` method extends the functionality of the `servlet.ProtectedServlet` superclass to indicate that when the HTTP client delegates to the gateway, it must in fact delegate to the compound principal $G|C$, "gateway quoting client."

```
public SDSIPrincipal getRequiredIssuer() {
    SDSIPrincipal client
        = new PseudoPrincipal("Your Identity Here");
        // Produce a pseudo-compound-principal, telling the client
        // where to fill in its own identity.
    return new Quoting(prover.getIdentityPublicKey(), client);
        // Construct G|?.
}
```

The `getResourceTag()` method likewise extends the superclass by describing how resources this application serves (views of email documents) map into Snowflake tags.

```
Tag getResourceTag() {
    // Describe each of the parameters of the request as different
    // tag components, to allow users maximum extensibility and
    // granularity in constructing delegations.
    SexpList messageSexp =
        new SexpList().append("messageId").append(getMessageId());
    SexpList mailSexp =
        new SexpList().append("mail").append(messageSexp);
    SexpList tagSexp =
        new SexpList().append("tag").append(mailSexp);
    Tag gatewayLevelTag = new Tag(tagSexp);
    Tag unionTag = gatewayLevelTag;
    if (serverTag!=null) {
        unionTag = unionTag.union(serverTag);
        // If the server has special requirements, also ask
        // client to meet server's requirements. This is only
        // a hint to save a trip and a separate proof, since
        // the server will be checking these requirements itself
        // and would reject an insufficiently-authorized request.
    }
    return unionTag;
}
```

Finally, when it needs to hand off its authority to the channel it is using, the gateway explicitly indicates the client it is working for. (It would be better to multiplex the RMI channels so that different RMI requests went over different logical channels, but this approach is a good start.)

```
// Make the statement M says K_CH ⇒ M|C.
Validity v = new Validity();
v.updateAfter(new Date(System.currentTimeMillis()+30000L));
Auth authCert = new Auth(q, ch, Tag.getTagStar(),
    true, v);
    // Unrestricted delegations are seldom used; here I choose an
    // arbitrary expiration time of 30 seconds for the delegation,
    // after which the gateway will automatically construct a new
    // one when the server rejects the expired delegation.
// The authCert object itself is the statement
// K_CH ⇒ M|C; the following signature
// adds the M says part that makes the statement ground
// truth.
SDSIPublicKey myPublicKey =
    prover.getPublicKeyForPrincipal((SDSIObject) q.getQuoter());
```

```
SDSIPrivateKey myPrivateKey =
    prover.getPrivateKeyForPublic(myPublicKey);
SDSISignature ss =
    new SDSISignature(authCert, myPrivateKey, myPublicKey);
SignedCertificate sc = new SignedCertificate(authCert, ss);
outProof = new SignedCertificateProof(sc, null);
prover.digestProof(outProof);
    // Hand the certificate to the Prover, who will use it
    // automatically when it needs to show the authority of the channel
    // over the RMI requests I am about to make.
```

These examples show how clients and servers of resources protected with Snowflake's security model access the tools in the `proof` package and related packages to establish their own authority and verify the authority of programs with which they communicate.

Bootstrapping.

Here are some mundane reminders about how the executable pieces are put together, to assist in repeating experiments.

```
cd  /snowflake
jdk-go
setenv CLASSPATH ‘make classpath‘
```

Insert `-Djava.compiler=NONE` on a command line to get more useful stack traces.

To start the proxy server that implements the client side of the Snowflake HTTP protocol, run

```
java jp.ProxyConfig certs-jon
```

To start a Snowflake HTTP server, including both the file servlet and the email gateway servlet, run

```
java jp.SecureServerConfig ’(nothing)’
```

To start a database, run

```
java sun.rmi.registry.RegistryImpl
java relational.SSHDatabase \
    ’(hash md5 |9sj+h6KmnTmPxoIiRB3V3g==|)’
```

To parse mail into the database, run

```
java relational.email.Mailbox mailbox remote
```

To start the servers used in the timing experiments, run

```
java servlet.SSLServerConfig -fourServers=true
java jp.SecureServerConfig -root /usr/local/apache/htdocs
java timingexp.TestRMIServer -publicKey certs-server/1.object
```

To run the timing experiments, run this command with an appropriate mode flag.

```
java timingexp.GenerateTestCases -mode=snowflake-signs \
    -runTests=true
```

The set of mode flags appear in GenerateTestCases. To figure out which experiment is relevant, start with the table that contains the numbers of interest. Use the index table in Appendix F to map the table to an experiment number. Look up the experiment number in the `several_experiments.m` matlab file, and see which `timedata`/ file the numbers are read from. The name of that file should indicate the GenerateTestCases mode that produced it. The hostname "shovel" in the filename means that the client (and any servers) were both on the same machine; the name "plow" means that the client was on machine plow, remote from the servers.

@author jonh@cs.dartmouth.edu

Package cal

The cal package is a calendar/appointment manager application based on Snowflake naming. Calendar queries are mapped into name resolution operations, so Snowflake name bindings can be used to hide distribution and administrative boundaries from this simple application. Similarly, a Union directory can be used to merge two calendar databases into one virtual calendar visible with this application.

Classes

Class **Event**

public class Event
extends java.lang.Object
implements java.io.Serializable

An Event is an (EventDescription, Occurrence) tuple.

An Event is the top type in the calendar schema; it binds timeless descriptions to specific times, so that descriptions can be reused with reference semantics. That is, you never need to copy a description, then update it in two places.

CONSTRUCTORS
public `Event`( )

Class **EventDescription**

public class EventDescription
extends java.lang.Object
implements java.io.Serializable

An EventDescription is the timeless description of an event. It can be reused for multiple occurrences, so that a single correction corrects every occurrence of the event in the calendar (reference semantics).

CONSTRUCTORS
public `EventDescription`( )

METHODS
public String `getDescription`( )
public String `getLocalTimeZoneName`( )
public void `setDescription`(java.lang.String s)
public void `setLocalTimeZoneName`(java.lang.String s)

Class **Importer**

public class Importer
extends java.rmi.server.UnicastRemoteObject
implements sf.Program, java.io.Serializable

CONSTRUCTORS
public `Importer`( )

METHODS
public long `parseDate`(java.lang.String cowendarDate, long defaultDate)

Usage Parse a date from the web calendar into a Unix-style seconds-since-Epoch dat.

public Object `run`(sf.Namespace root)

Usage The command-line interface to the Importer. Specify as arguments the URL of the calendar to import from, and the Snowflake name of the Container to import the events into.

public static Vector `split`(char sep, java.lang.String s)

Usage Kind of like the Perl split() function. This belongs in the `Tools` package.

Class **Importer.CommentSkipper**

public class Importer.CommentSkipper
extends java.io.FilterReader

An I/O filter class that skips lines starting with #. The (Dartmouth) web calendar export files include such comment lines.

CONSTRUCTORS

public `Importer.CommentSkipper`(cal.Importer this$0, java.io.Reader in)

METHODS

public boolean `isCR`(int ch)
public int `read`( )
public int `read`(char []cbuf, int off, int len)

Class **Occurrence**

public class Occurrence
extends java.lang.Object
implements java.io.Serializable

An Occurrence is the temporal manifestation of an EventDescription. EventDescriptions may occur weekly, in which case one Occurrence represents each week. An Occurrence represents one start and one ending.

CONSTRUCTORS

public `Occurrence`( )

METHODS

public long `getEndTime`( )

Usage Get the end time in seconds since the Unix epoch.

public long `getStartTime`( )

Usage Get the start time in seconds since the Unix epoch.

public boolean `getTimesMeaningful`( )

Usage Learn whether the times are significant, or whether this occurrence is only specified at "day-long" resolution.

public void `setEndTime`(long t)

Usage Set the end time in seconds since the Unix epoch.

public void `setStartTime`(long t)

Usage Set the start time in seconds since the Unix epoch.

public void `setTimesMeaningful`(boolean m)

Usage Specify whether the times are significant, or whether this occurrence is only specified at "day-long" resolution.

Class **Query**

public abstract class Query **extends** java.rmi.server.UnicastRemoteObject **implements** sf.Namespace

A Query is a Namespace that contains Events that match a given query. A namespace above this accepts lookups where the string specifies the query; the result of the lookup is a dynamically-generated instance of this class that provides the pool of events that match the lookup query.

This class is abstract. Subclasses specify particular types of queries that they implement.

CONSTRUCTORS
public `Query`( )

METHODS
public void `bind`(java.lang.String name, java.lang.Object o)
public boolean `completeList`( )
public Vector `listAllNames`( )
public Object `lookupName`(java.lang.String name)
public Object `lookupPath`(java.lang.String name)
public Object `lookupPath`(java.util.Vector path, int cur)
public int `version`( )

Class **textview**

public class textview **extends** java.rmi.server.UnicastRemoteObject **implements** sf.Program, java.io.Serializable

A text-based tool for inspecting a calendar (a Namespace full of Events).

CONSTRUCTORS
public `textview`( )

METHODS
public Object `run`(sf.Namespace root)

Usage Run the tool from the Snowflake shell command line, specifying the path to the Namespace containing the Events to be viewed. The Namespace is often a Query on some other calendar. If unspecified, this program looks for a calendar at `/cal`.

Class **TimeQuery**

public class TimeQuery **extends** cal.Query **implements** sf.Namespace, sf.Program, java.io.Serializable

TimeQuery is a Query that matches events that overlap a given time interval.

CONSTRUCTORS
public `TimeQuery`( )

Methods

public Object `run`(sf.Namespace root)

Usage Manually instantiate a timequery object from the Snowflake shell, specifying the time range parameters.

Package gb

The gb package is an awt-based graphical browser for Snowflake namespaces.

Interfaces

Interface **GUISelector**

public interface GUISelector

A GUISelector is a tool that knows how to find an already-open window in which to display a specific resource.

METHODS

public Window `select`(sf.Namespace root, java.lang.String path, gb.Browser browser)

Classes

Class **Browser**

public class Browser
extends java.rmi.server.UnicastRemoteObject
implements sf.Namespace, sf.Program, java.io.Serializable

A Snowflake shell program that creates a graphical Namespace browser.

CONSTRUCTORS

public `Browser`( )

METHODS

public void `bind`(java.lang.String name, java.lang.Object o)
public boolean `completeList`( )
public Vector `listAllNames`( )
public synchronized void `listClosed`(java.lang.String path)

Usage Remove a window from the list of open windows.

Parameters
`path` - The Snowflake name for the resource the window is displaying.

public Object `lookupName`(java.lang.String name)
public Object `lookupPath`(java.lang.String name)
public Object `lookupPath`(java.util.Vector path, int cur)
public void `openList`(java.lang.String path)

Usage Open a new window showing a Namespace resource.

Parameters
`path` - specifies the Snowflake path to the resource to display.

public void `openList`(java.lang.String path, java.awt.Point location)

Usage Open a new window showing a Namespace resource.

Parameters
`path` - specifies the Snowflake path to the resource to display.
`location` - specifies the location for the display on the screen.

public synchronized void `openListImpl`(gb.Browser.OpenTask task)

Usage Open any windows queued for opening. A thread watches the queue of windows waiting to be opened and calls this method to do the work.

public Object `run`(sf.Namespace root)

Usage Create a browser from the Snowflake shell.

public int `version`( )

Class **Browser.DefaultGUISelector**

public class Browser.DefaultGUISelector
extends java.lang.Object
implements GUISelector

This DefaultGUISelector picks a GUI based on the object we're viewing. It is used to map a request to an already-open window, where appropriate, or to find an appropriate GUI tool for the interface of the resource being opened.

This simple implementation knows how to display Namespaces, `mail.Mailbox`es, and `mail.MSU` ("mail storage units").

Constructors
public `Browser.DefaultGUISelector`(gb.Browser this$0)

Methods
public Window `select`(sf.Namespace root, java.lang.String path, gb.Browser browser)

Class **Browser.OpenTask**

public class Browser.OpenTask
extends java.lang.Object

An OpenTask remembers a task that we intend to do while it sits in a queue, waiting for the Opener thread to get a chance to service it.

Class **gbinput**

public class gbinput
extends java.awt.Panel

A one-line text input field with emacs-like keystroke editing commands. Used when instantiating a Shell in graphical mode.

Constructors
public `gbinput`( )

Usage Instantiate an 80-character-wide input field.

public `gbinput`(int width)

Usage Instantiate an input field.

Parameters
`width` - width of field, in number of characters.

Methods
public void `processEnter`( )

Usage The user typed `enter`; send the command to the inputBuffer where it waits to get read out by readLine().

public String `readLine`( )

Usage Read out a single typed command. When the user types a command, it sits in a queue until read out with a call to this method.

public void `scrollHistory`(int dist)

Usage Scroll up and down in the history of entered commands. Called when the user types ∧P or ∧N.

Class **GBList**

public class GBList
extends java.awt.Component
implements java.awt.ItemSelectable

GBList is a Graphical Browser List view. It displays the names bound in a Namespace in an awt List window. It is supplied with names from a Namespace by `NSListPanel`.

Constructors

public `GBList`( )
public `GBList`(int i)

Methods

public synchronized void `addItem`(java.lang.Object itemKey, java.lang.String itemName, int index)
public void `addItem`(java.lang.String itemName)
public synchronized void `addItemListener`(java.awt.event.ItemListener itemListener)
public String `getItem`(int i)
public int `getItemCount`( )
public Object `getItemKey`(int index)
public String `getItemName`(int index)
public Dimension `getMinimumSize`( )
public Dimension `getPreferredSize`( )
public Object `getSelectedObjects`( )
public synchronized void `paint`(java.awt.Graphics g)
protected void `processEvent`(java.awt.AWTEvent ev)
protected void `processItemEvent`(java.awt.event.ItemEvent e)
public synchronized void `removeAllItems`( )
public synchronized void `removeItem`(java.lang.Object itemKey)
public synchronized void `removeItemListener`(java.awt.event.ItemListener itemListener)

Class **gboutput**

public class gboutput
extends java.awt.Panel

A GUI window that displays scrolling output text.

Constructors

public `gboutput`( )

Usage Create a 24x80 scrolling output window.

public `gboutput`(int rows, int cols)

Usage Create a scrolling output window.

Parameters
- `rows` - number of rows of text to display
- `cols` - number of columns of characters to display

METHODS

public OutputStream `getOutputStream`( )

Usage Get an OutputStream for this window. Any text written to the output stream will appear in this window. A shell binds the OutputStream returned by this method to `/streams/stdout` in a Snowflake namespace, to cause all Snowflake programs to send their output to the scrolling text window.

Class **NSListPanel**

public class NSListPanel **extends** java.awt.Panel

An NSListPanel connects a Snowflake Namespace to a `GBList` GUI display of names. It monitors the Namespace for changes using the NamespaceListener interface to keep the GUI updated. It also catches `ItemEvent`s from the GBList that represent clicks on names, and opens a new window to display the underlying resource.

CONSTRUCTORS

public `NSListPanel`(sf.Namespace rtparam, java.lang.String nspathparam, gb.Browser brparam)

Usage Create a new GUI Namespace display.

Parameters
- `rtparam` - root of a Snowflake namespace
- `nspathparam` - path from `rtparam` that specifies the Namespace to display.
- `brparam` - the `Browser` to use when opening new resources.

METHODS

public Insets `getInsets`( )

Class **NSListPanel.NListen**

public class NSListPanel.NListen **extends** sf.NamespaceListenerAdapter

NListen keeps tabs on Namespace in case it changes, and updates the GUI accordingly.

SERIALIZABLE FIELDS

private final NSListPanel `this$0`

CONSTRUCTORS

public `NSListPanel.NListen`(gb.NSListPanel this$0, gb.NSListPanel panel)

METHODS

public void `namespaceEvent`(sf.NamespaceEvent ev)

Usage The Namespace has changed. Pass it to my outer class.

Package Icee

This is the "icee" process checkpointer, designed especially to provide persistence for Java, which relies on more sophisticated state than the typical scientific program. Icee should run on Solaris 2.5 and Solaris 2.6.

To build: `(cd Icee; make)`

If you are on a Solaris 2.5 system, you may need to use: `(cd Icee; make depend; make)`

To try it out, from this directory, the one containing Icee/, do:

```
setenv CLASSPATH Icee:$CLASSPATH
    setenv LD_LIBRARY_PATH Icee:$LD_LIBRARY_PATH
        # (modify the above statements as necessary if you use sh)
    Icee/icee Icee.Auto -verbose -period=5 Icee.Demo
```

After ten seconds or so (once you've seen a checkpoint), hit ∧C, and try:

`Icee/icee -recover`

The paper is at: `http://www.cs.dartmouth.edu/ jonh/research/pjw3/`

@author Jon Howell jonh@cs.dartmouth.edu

Interfaces

Interface **Auto.Callback**

public static interface Auto.Callback

An object implements Callback to receive notification that a checkpoint recovery has occurred.

METHODS

public void `recovered`( )

Usage This method is called whenever the checkpointer recovers a JVM from a failure. Use it to catch such events and reopen state (such as network connections) that are not automatically re-established by Icee.

Classes

Class **Auto**

public class Auto **extends** java.lang.Thread

A class that "wraps" classes you really want to run, first starting a daemon thread to periodically invoke the checkpointing process. It "wraps" another class in the sense that it is a minor syntactic change to the Unix command line:

```
java myotherclass myargs
```

becomes

```
java Auto -autoargs myotherclass myargs
```

(currently the second 'java' needs to be 'icee', since icee cannot be loaded dynamically.)

CONSTRUCTORS

public `Auto`(boolean verbose, int period)

METHODS

public static void `main`(java.lang.String []args)
public static void `registerCallback`(Icee.Auto.Callback c)
public void `run`( )

Class **Control**

public class Control **extends** java.lang.Object

The Java native class that provides the interface to the icee checkpointing service.

CONSTRUCTORS

public `Control`( )

METHODS

public static native int `doCheckpoint`( )

Usage Call this method to take a checkpoint. Recovery always appears to happen "during" a checkpoint (since that's exactly the state of the system when it was saved), so it will be at the return from this call that your program should test to see if we just recovered and need to perform any special cleanup. The return value indicates whether the program has just recovered.

Returns value of 0 if all is normal, value of 1 if we just recovered from a checkpoint.

public static native void `doRestore`( )

Usage The hook to restore an existing checkpoint from Java code. This is like a longjmp; it never returns.

Returns on success, doesn't return (restored process sees its doCheckpoint() return a 1 value).

Class **Demo**

public class Demo
extends java.lang.Object

A class that shows the checkpointer "in action." Invoke with:
`icee Icee.Auto -verbose -period=5 Icee.Demo`

Then hit $\wedge$C, and restart with: `icee -recover`

CONSTRUCTORS

public `Demo`( )

METHODS

public static void `main`(java.lang.String []args)

Package ide

The ide package contains Tools for working with Java code inside Snowflake. Included are tools to convince `javac` to manipulate source and class files through the Snowflake interface, as well as a ClassLoader to load classes from Snowflake Namespaces. This package also contains the textual Shell for accessing Snowflake `sf.Program`s, and the RemoteInputStream and RemoteOutputStream interfaces and implementations for accessing Java-style streams across RMI.

Interfaces

Interface **RemoteInputStream**

public interface RemoteInputStream **extends** java.rmi.Remote

A Remote version of InputStream, to allow `java.io.InputStreams` to be passed across JVMs. Particularly useful because it lets us bind InputStreams into Snowflake Namespaces.

Methods

public int `available`( )
public void `close`( )
public int `read`( )
public RISReturn `read`(int max)
public void `reset`( )
public long `skip`(long n)

Interface **RemoteOutputStream**

public interface RemoteOutputStream **extends** java.rmi.Remote

A Remote version of OutputStream, to allow `java.io.OutputStreams` to be passed across JVMs. Particularly useful because it lets us bind OutputStreams into Snowflake Namespaces.

Methods

public void `close`( )
public void `flush`( )
public void `write`(byte []b)
public void `write`(byte []b, int off, int len)
public void `write`(int b)

Classes

Class **AddOMatic**

public class AddOMatic **extends** java.rmi.server.UnicastRemoteObject **implements** sf.Program, java.io.Serializable

A very simple program to demonstrate the Program, RemoteInputStream, and RemoteOutputStream interfaces. It reads lines from the Snowflake standard input stream and prints a running total of their numeric values.

Constructors

public `AddOMatic`( )

Methods

public Object `run`(sf.Namespace root)

Usage The Snowflake command-line (Program) interface. Reads lines, prints sums.

Class **ClassDependency**

public class ClassDependency **extends** java.lang.Object

Search a list of class files supplied on the input (using ide.ClassFile to examine their bytecodes directly), looking for any that reference a given CONSTANT_Class in their constant tables.

CONSTRUCTORS
public `ClassDependency`( )

METHODS
public static void `main`(java.lang.String []args)

Class **CLSnooper**

public class CLSnooper **extends** java.lang.ClassLoader

A CLSnooper was an attack at the "class evolution" problem. It was designed to be attached to a ContainerServer to allow one to add new objects of a revised class without discarding the old objects. Unfortunately, since even my interfaces were changing rapidly at this time, not enough of the system was stable enough to allow the new and old objects to communicate usefully.

CONSTRUCTORS
public `CLSnooper`( )

Usage Create a snooping classloader. It defines several classes as precious, indicating those that the CLSnooper shouldn't try to load on its own lest it confuse the JVM.

METHODS
protected Class `loadClass`(java.lang.String name, boolean resolve)

Usage Attempt to resolve almost all classes (except those marked precious) by myself. Leave very little up to the JVM, so that when this ClassLoader is replaced with a fresh one, almost all classes get reloaded with new versions.

Class **CLSnooper.IndentStream**

public class CLSnooper.IndentStream **extends** java.lang.Object

An indenting output stream that is superseded by a niftier class in the Tools package.

CONSTRUCTORS
public `CLSnooper.IndentStream`(ide.CLSnooper this$0)
public `CLSnooper.IndentStream`(ide.CLSnooper this$0, java.io.PrintStream baseStream)

METHODS
public void `indent`( )

public void `outdent`( )
public void `println`(java.lang.String line)

Class **CopyClass**

public class CopyClass
extends java.lang.Object

A test of ide.ClassFile. This should open a .class file as a ClassFile, then be able to write it out to a new .class file, without changing the semantics of the result. (Eventually a different program will be able to modify/instrument the class file before writing it out.)
@created Mon Oct 19 10:13:04 EDT 1998

Constructors
public `CopyClass`( )

Methods
public static void `main`(java.lang.String []args)

Class **javac**

public class javac
extends java.rmi.server.UnicastRemoteObject
implements sf.Program, java.io.Serializable

A wrapper for Sun's `javac` class to retrain it to retrieve source code and classes from a Snowflake namespace rather than the Unix filesystem.

Constructors
public `javac`( )

Methods
public Object `run`(sf.Namespace root)
Usage The Program (Snowflake shell command line) interface to the Java compiler. Javac's calls to FileInputStream and FileOutputStream are redirected (using class file rewriting) to `SFFileInputStream` and `SFFileOutputStream`, which talk to the Snowflake namespace.

Class **loader**

public class loader
extends java.rmi.server.UnicastRemoteObject
implements sf.Program, java.io.Serializable

A Program (Snowflake shell) interface to SFClassLoader, to enable the user to explicitly load a Java class from a Snowflake Namespace.

Constructors
public `loader`( )

Methods
public Object `run`(sf.Namespace root)
Usage The Program implementation.

Class **RemoteHello**

public class RemoteHello
extends java.lang.Object

Test the RemoteOutputStream by sending "hello" down it.

CONSTRUCTORS
public `RemoteHello`( )

METHODS
public static void `hello`(java.lang.String []args, ide.RemoteOutputStream os)

Class **RISIn**

public class RISIn
extends java.rmi.server.UnicastRemoteObject
implements RemoteInputStream

This class adapts an InputStream to the RemoteInputStream interface. Use it to export a local InputStream as a distributed (Remote) Snowflake resource.

@classConcise true

CONSTRUCTORS
public `RISIn`(java.io.InputStream is)

Class **RISOut**

public class RISOut
extends java.io.InputStream

This class adapts a RemoteInputStream to the InputStream "interface" (abstract class — yuk). Use it to pass a RemoteInputStream to existing Java code that expects a java.io.InputStream.

@classConcise true

CONSTRUCTORS
public `RISOut`(ide.RemoteInputStream ris)

Class **RISReturn**

public class RISReturn
extends java.lang.Object
implements java.io.Serializable

RemoteInputStream's bulk `read()` interface is a little different than that of InputStream. In InputStream.read, you pass in an array by reference, and let the read() method populate it. Filling an array by Remote reference is a pretty bad idea; and in any case, arrays aren't Remote, so using the same interface would involve sending a long, empty array across the net, only to have the (partially-)populated array returned as a return value.

Instead, this class acts as a "packet" to carry the return value from

`RemoteInputStream.read`. The argument to read is an integer specifying the maximum number of bytes to read. The result is this packet, carrying the bytes plus an `rc` value used to indicate an EOF condition.

SERIALIZABLE FIELDS

public int `rc`
- -1 => EOF, else rc == b.length

public byte `b`
- data read from stream

CONSTRUCTORS

public `RISReturn`( )

Class **ROSIn**

public class ROSIn
extends java.io.OutputStream

The input-end of a RemoteOutputStream pipe; looks like a local OutputStream. Use it to pass a Snowflake RemoteOutputStream to existing Java code that expects a conventional java.io.OutputStream.

@classConcise true

CONSTRUCTORS

public `ROSIn`(ide.RemoteOutputStream ros)

Class **ROSOut**

public class ROSOut
extends java.rmi.server.UnicastRemoteObject
implements RemoteOutputStream

This class adapts a java.io.OutputStream to the distributed (Remote) Snowflake RemoteOutputStream interface. Use it to export a Java OutputStream resource as a Snowflake resource.

@classConcise true

CONSTRUCTORS

public `ROSOut`(java.io.OutputStream os)

Class **Set**

public class Set
extends java.util.Hashtable

A Set from before java.util.Set appeared in JDK1.2.

@classConcise true
@deprecated since JDK1.2 finally supports this functionality.

CONSTRUCTORS

public `Set`( )

Class **SFClassLoader**

public class SFClassLoader
extends java.lang.ClassLoader

A ClassLoader that reads Java classes from Snowflake resources in a Snowflake namespace.

CONSTRUCTORS

public `SFClassLoader`(sf.Namespace root)

Usage Construct a ClassLoader, specifying the root namespace from which to begin searching for Java class files.

METHODS

protected Class `loadClass`(java.lang.String name, boolean resolve)

Class **SFFileInputStream**

public class SFFileInputStream
extends ide.RISOut

A replacement for java.io.FileInputStream that opens files from the Snowflake namespace. Can be brute-force substituted into classes that expect FileInputStreams using `TweakClass`.

CONSTRUCTORS

public `SFFileInputStream`(java.io.File filename)

Usage compatibility constructor to match calls to FileInputStream methods

public `SFFileInputStream`(java.lang.String path)

Usage compatibility constructor to match calls to FileInputStream methods

Class **SFFileOutputStream**

public class SFFileOutputStream
extends ide.ROSIn

A replacement for java.io.FileOutputStream that opens files from the Snowflake namespace. Can be brute-force substituted into classes that expect FileOutputStreams using `TweakClass`.

CONSTRUCTORS

public `SFFileOutputStream`(java.io.File filename)

Usage compatibility constructor to match calls to FileOutputStream methods

public `SFFileOutputStream`(java.lang.String path)

Usage compatibility constructor to match calls to FileOutputStream methods

Class **Shell**

public class Shell
extends java.lang.Object

A text-based command-line shell. It accepts input commands, looks them up in the user's Snowflake root Namespace under `/cmd`, executes them with the remaining arguments bound into the subprogram's local namespace, and awaits another command.

This instantiation is substantially different than `jonh.Shell`; the latter, for example, is the only one that uses an awt window to display output and collect input. This class also has lost its Program interface, for some reason. Strange, since there's no reason one might not want to invoke shells recursively!

CONSTRUCTORS

public `Shell`( )

METHODS

public static void `main`(java.lang.String []args)

Usage Create a shell that uses System.in and System.out for I/O streams from the Unix command line. This shell includes some default bindings for the early `sf.sec` and `sf.rsec` Snowflake security model.

public static void `shell`( )

Usage The main Program loop of the shell. Retrieves its I/O streams from the Snowflake namespace, and loops processing commands.

public static Vector `split`(java.lang.String s)

Usage Perl-like split() function separates words on a command line.

Class **Split**

public class Split
extends java.lang.Object

Yet another implementation of a simple perl-like split() function.

@todo belongs in the Tools package.

CONSTRUCTORS

public `Split`( )

METHODS

public static Vector `split`(java.lang.String input, char delimiter)

Class **TweakClass**

public class TweakClass
extends java.lang.Object

This program opens a .class file as a ClassFile, then be able to write it out to a new .class file that references class B instead of class A. It is a static implementation of class reference substitution. One idea was to do this on the fly in a classloader, so that all Java programs could be transparently rewritten (for example, to get javac to use SFFileInputStreams). Unfortunately, that approach didn't pan out, so the `javac` case is handled by manually invoking this TweakClass program to translate the necessary parts of `javac`.

@created Mon Oct 19 11:22:26 EDT 1998

CONSTRUCTORS

public `TweakClass`( )

METHODS

public static void `main`(java.lang.String []args)

Usage Unix command-line interface.

public void `realMain`(java.lang.String []args)

Usage Takes four arguments:

- **inClass** Class to tweak
- **outClass** Name of output class
- **classA** Class reference in inClass to change
- **classB** What to replaces references to `classA` with

Class **TweakClass.StoppyOutputStream**

public class TweakClass.StoppyOutputStream
extends java.io.FilterOutputStream

Stoppy is a debugging tool. When a rewritten class file doesn't work right, you can rewrite the class file with a null change, which should generate the same classfile. If it does not, then there's a bug in the input or output code of Classes.ClassFile. To find it, 'od -Ax -tx1' the class files, and diff them. Figure out what hex offset the thing screws up on, and set that in the written== test in write(int b). Then put a breakpoint there in the debugger, pop up a bit on the stack, and you'll know which scumsucking cretin routine is doing the screwing up.

(Don't forget to uncomment the code below that actually uses Mister Stoppy.)

@classConcise true

CONSTRUCTORS

public `TweakClass.StoppyOutputStream`(ide.TweakClass this$0, java.io.OutputStream out)

Package ide.Classes

This package "introspects" on a class file. Its components reflect all of the parts of a class file, and are used by classes such as `ide.TweakClass` that want to rewrite a class file.

The primary class is `ClassFile`; it loads a `.class` file and parses it into objects represented by other classes in this package.

I have omitted these classes from the manual because they are largely mechanical details, reifying various components of the Java Virtual Machine specification.

Package jp

This package is the web proxy that implements client side of the Snowflake HTTP protocol. Its primary classes are the `SfUserAgent` that implements the protocol itself and the `PrincipalManager` that provides the user interface. It is described in my dissertation in Section 10.3.

Interfaces

Interface **RequestStates**

public interface RequestStates

The SfUserAgent's protocol is modeled as a small state machine. These are its states.

FIELDS

public static final int `TRY_IDENTICAL`
- First authorization attempt is to see if this request is identical to one we've sent before.

public static final int `TRY_MAC`
- The next-best thing after an identical request is a connection to a server that we share a MAC secret with; the request can be very quickly authenticated using a simple signature.

public static final int `TRY_HINT`
- See if we have sent a request before to a prefix of this URL, and it required Snowflake authorization. If so, it is prudent to try using the same authority over this request that looks like a sub-request, to save a round trip.

public static final int `TRY_NOTHING`
- We have no useful clues. Try sending the request without Snowflake authorization, and if Sf auth is required, the server will demand it.

public static final int `SEND_REQUEST`
- Having decided what authorization mechanism we're going to use on this pass, send the request to the server.

public static final int `REQ_DONE`
- The request has been answered, and we have no way to improve on it even if the answer was "401 Unauthorized," so return the result to the client.

Interface **SfHttpProtocol**

public interface SfHttpProtocol

The definition of constants in the Snowflake version of the HTTP Authorization protocol.

FIELDS

public static final String `HTTP_AUTH_CHALLENGE`
- Sent in an HTTP response (how's that for confusing? It's because the server is challenging the client) to demand HTTP authorization of the client.

public static final String `HTTP_AUTH_RESPONSE`
- The client responds to the HTTP_AUTH_CHALLENGE in its second request, transmitting its proof of authority in the value of this header.

public static final String `SNOWFLAKEPROOF`
- The identifier for the Snowflake authorization method, supplied as the first word after the HTTP_AUTH_CHALLENGE.

public static final String `AUTHORIZECLIENT`

- Means the client needs to authorize itself; service issuer and minimum tag are supplied as arguments in `SERVICEISSUER` and `MINIMUMTAG` headers.

public static final String `IDENTIFYCLIENT`

- A challenge that tells the client to name a principal it wishes to use as its identity. (The client doesn't have to prove anything about that principal yet.) This challenge is presented by a gateway trying to learn on whose behalf it should operate.

public static final String `AUTHORIZEPROXY`

- Means the client needs to give the proxy (server in this transaction) authority to perform the transaction. Ultimate service issuer, minimum tag, and proxy principal who will be quoting the issuer are supplied.

  A client that blindly responds to this challenge without considering the trustworthiness of the challenging gateway risks falling for a man-in-the-middle attack. The client should also consider the strength of the delegation requested to avoid placing too much trust in the gateway.

  Note that "challenge" is not the best word for this demand; the gateway (or "proxy," as I sometimes refer to the gateway in this code) is actually requesting that the client delegate some authority to it.

public static final String `SERVICEISSUER`

- An extra header that carries information about the demands of an `HTTP_AUTH_CHALLENGE`. This header specifies the issuer that the proof must show the request speaks for.

public static final String `MINIMUMTAG`

- An extra header that carries information about the demands of an `HTTP_AUTH_CHALLENGE`. This header specifies the minimum restriction set that contains the request that inspired the challenge.

public static final String `PROXYPRINCIPAL`

- An extra header that describes the principal that must speak for the client (and eventually the resource server); presented with AUTHORIZEPROXY challenges.

public static final String `AUXILLIARYFACTS`

- An extra header, now unused. Once designed to carry extra facts either direction to be deposited in the recipient's Prover.

public static final String `CLIENTIDENTITY`

- An extra header indicating the client's identity in response to an `IDENTIFYCLIENT` challenge.

public static final String `REQUESTMAC`

- The client sends this extra header, with an argument giving its public key, to request that the server generate a secret MAC (Message Authentication Code) and return it to the client encrypted with the client's public key.

  The server should be careful to ensure that any REQUESTMAC header belongs to the signed text of a request, since it assumes a delegation from the MAC itself to the signer of the request. An adversary could otherwise inject a request for a MAC into a message, and steal the client's authority.

  The MAC protocol is something I brewed up, and it depends on secrecy, which my logic says little about. It would be prudent to study this protocol further or substitute a better-known protocol before trying to deploy this protocol in production.

public static final String `ENCRYPTEDMAC`
- The server sends the encrypted MAC back to the client in this header.

public static final String `DOCFORSERVERNAME`
- A server sends to the client in this extra header a proof that the document content (bytes following the headers and first blank line) of this message speak for a symbolic name bound in the server's secure SPKI namespace. The document is represented in the proof by its SPKI `sdsi.ObjectHash`. This a simple form of server authentication; note that it does *not* verify the authority of the headers returned by the server, so (for example) it does not protect the MAC protocol from MITM attacks.

Classes

Class **DigestInputStream**

public class DigestInputStream **extends** java.io.FilterInputStream

A DigestInputStream is a convenience filter for taking the MD5 (or other digest) of a data stream as it flows from source to sink, and ensuring that it matches an expected digest (hash) value.

@classConcise true
@author jonh@cs.dartmouth.edu

Constructors

public `DigestInputStream`(java.io.InputStream stream, sdsi.Hash expectedHash)

Usage Install a digest-checker on a stream. @param stream the source of bytes to digest @param expectedHash the expected hash value of the complete stream

Methods

public void `close`( )

Usage When the stream is closed, this overriding method will automatically check the stream's digest.

Exceptions

`jp.DigestStreamException` - if the hashes do not match.

Class **ForwardedHttpServletRequest**

public class ForwardedHttpServletRequest
extends jp.ForwardedServletRequest
implements javax.servlet.http.HttpServletRequest

Convenince class to implement the `javax.servlet.http.HttpServletRequest` interface to allow a Server to build a "replacement" request based on an original request, but with certain changes overlaid. Think of it as a way to implement "union mount" for a Request.

All of the method implementations in this class forward to the corresponding methods in the prototype object. A subclass need only override the methods it wishes to interpose upon.

@classConcise true
@author Jon Howell jonh@cs.dartmouth.edu

Class **ForwardedServletRequest**

public class ForwardedServletRequest
extends java.lang.Object
implements javax.servlet.ServletRequest

Convenince class to implement the `javax.servlet.http.ServletRequest` interface to allow a server to build a "replacement" request based on an original request, but with certain changes overlaid.

All of the method implementations in this class forward to the corresponding methods in the prototype object.

@classConcise true
@author Jon Howell jonh@cs.dartmouth.edu

Class **History**

public class History
extends java.lang.Object

A History object maintains a list of PageHistory objects, used to implement the history list in the PrincipalManager user interface.

CONSTRUCTORS

public `History`(int size)

Usage Create a History list.

Parameters

`size` - maximum number of history entries to maintain.

METHODS

public void `addHistory`(jp.PageHistory h)

Usage Insert a page reference in the history list. The oldest page is discarded.

public PageHistory `findPage`(java.lang.String url)

Usage Look up a PageHistory object by URL. Used by PrincipalManager when trying to map a clicked URL back to the PageHistory object that carries references to the Snowflake authorization information used when the page was accessed.

public Iterator `iterator`( )

Usage Retrieve an iterator that returns `PageHistory` objects in reverse chronological order (newest first).

Class **IncomingResponse**

public class IncomingResponse
extends com.mortbay.HTTP.HttpHeader

This class represents an incoming HTTP response, whose result code and headers have been extracted. It's very different than com.mortbay.HTTP.HttpResponse, which is a response being constructed to send outbound. This class doesn't (yet) have any useful write operations, it's only a way to examine that incoming header. It is being designed as part of a proxy, which reads the header, makes a decision, then (sometimes) passes the *original stream* on to the client, not this header. I imagine someday I could extend this to actually act like an HttpResponse, so that the proxy could send it back out the same way it does a fresh HttpResponse.

CONSTRUCTORS

public `IncomingResponse`(java.io.InputStream is)

Usage Parse a stream into an IncomingResponse object, ready to return headers or the content stream.

public `IncomingResponse`(java.io.InputStream is, boolean rewindable)

Usage Parse a stream into an IncomingResponse object, ready to return headers or the content stream.

public `IncomingResponse`(java.io.InputStream is, boolean rewindable, boolean closeInputStream)

Usage Parse a stream into an IncomingResponse object, ready to return headers or the content stream.

Parameters

`rewindable` - - if false, some overhead is saved by not setting up a RecordingInputStream to capture the protocol line & headers for later replay. Generally, rewindable = true for proxies that need to inspect the stream but then replay it verbatim later; else it should be false.

`closeInputStream` - - if true, the `is` argument is closed when close() is called on this object.

METHODS

public void `attachDocumentVerifier`(sdsi.ObjectHash objHash)

Usage Attach a message digest verifier to this stream, so that if, when the stream is fully read out, its digest does not match, the stream throws an IOException.

public void `close`( )

Usage Close the content stream and the input stream that this object was parsed from.

public int `getCode`( )

Usage Return the numeric result code of the response, e.g., 100 (OK).

public InputStream `getContentStream`( )

Usage Return a stream containing the content (everything past the headers).

public String `getDesc`( )

Usage Return the string descriptor that follows the numeric result code.

public InputStream `getReplayStream`( )

Usage Return a stream containing the entire response: protocol line, headers, separator line, content.

public String `getResponseLine`( )

Usage Return the response line of the response: the first line, with the numeric result code.

public void `passAlongResponse`(com.mortbay.HTTP.HttpResponse outgoingResponse)

Usage A proxy uses this method to pass this incoming response out to a calling client using an outgoing HttpResponse object. If we had an exception parsing the input stream, then we play out the input verbatim. Otherwise, we play out this object as a header block followed by the rest of the input stream. That allows any changes made to the headers to show up at the client.

public void `write`(java.io.OutputStream out)

Usage Write this response onto the specified OutputStream. This method only writes out the response line and the headers; it does not copy the content. Hmm, maybe it does not even write out the response line.

Class **MacGuy**

public class MacGuy **extends** java.lang.Object

A little wrapper class to wrap up a MAC along with its precomputed hash; used to store MAC info together in a HashMap in `SfUserAgent`.

Class **PageHistory**

public class PageHistory
extends java.lang.Object

A PageHistory contains the state associated with visiting a web page. It includes the Snowflake (HTTP with signed requests protocol) authorization information, useful for delegating authority over the page to another.

Fields

public String `url`
- The URL of the visited page.

public String `title`
- HTML document title, if known. (currently, I'm not parsing out of the response stream.)

public int `snowflakeStatus`
- Indication of whether Snowflake authorization was used for this document, and its outcome.

public Proof `sfProof`
- The Snowflake proof used to access this document. NULL when snowflakeStatus!=SF_SUCCESS.

public static final int `SF_NOSFAUTH`
- loading this page did not require a Snowflake proof authorization

public static final int `SF_SFUNAUTH`
- this page did require a Snowflake proof, but we couldn't produce it.

public static final int `SF_SUCCESS`
- this page required a Snowflake proof, and we supplied it

Constructors

public `PageHistory`( )

Class **PrincipalManager**

public class PrincipalManager
extends javax.servlet.http.HttpServlet

This class implements the user interface to the SfUserAgent. It is a servlet that managing keys, delegations, and name bindings via the web browser.

@author jonh@cs.dartmouth.edu

Constructors

public `PrincipalManager`(proof.Prover2 prover, jp.History history)

Usage A principal manager is instantiated by the SfUserAgent, and given references to the SfUserAgent's prover and page view history.

METHODS

public void `doGet`(javax.servlet.http.HttpServletRequest request, javax.servlet.http.HttpServletResponse response)

Usage A request directed at the PrincipalManager is delivered here by the servlet mechanism.

public static void `initialize`( )

Usage The standard servlet initialization method.

Class **ProxyConfig**

public class ProxyConfig
extends com.mortbay.HTTP.Configure.BaseConfiguration

A Jetty configuration class that sets up a proxy with an `SfUserAgent` installed to process all outgoing requests.

CONSTRUCTORS

public `ProxyConfig`(Tools.Options opts)

Usage Create a configuration bound at the given host address and port.

METHODS

public static void `main`(java.lang.String []args)

Usage Start the SfUserAgent proxy server.

Class **SecureServerConfig**

public class SecureServerConfig
extends com.mortbay.HTTP.Configure.BaseConfiguration

A Jetty configuration class that sets up a servlet server, the secure (Snowflake-HTTP) file servlet, and the secure mail gateway.

CONSTRUCTORS

public `SecureServerConfig`(Tools.Options opts)

Usage Create a configuration bound at the given host address and port.

METHODS

public Class `listenerClasses`( )

Usage Override a default mortbay method to supply `servlet.NaglessListener` listeners to handle requests.

public static void `main`(java.lang.String []args)

Usage Start the servlet server and servlets from the Unix command line.

Class **SfUserAgent**

public class SfUserAgent **extends** com.mortbay.HTTP.Handler.NullHandler **implements** SfHttpProtocol, RequestStates

This handler is called to manage requests on the proxy port

The Sf HTTP Authorization protocol is based on the HTTP "Authentication" spec in RFC 2617. ftp://ftp.isi.edu/in-notes/rfc2617.txt

It's called a SfUserAgent to represent the fact that it is trying to act like part of the user's web browser. (It belongs on the same host, for example.) And the notion of "proxy" in Snowflake has to do with protocol translation in the middle of a transaction somewhere. This use of HTTP proxying is meant to be the endpoint of a transaction, as close as we can get to the user.

@todo Ensure that it's always the same user accessing this proxy, perhaps by using `identd` on localhost.
@todo turn history-tracking stuff into a second handler layer that's independent of the authenticating proxy. (Would that require a second snoop-'n'-parse of the incoming headers? yuk!)
@author jonh@cs.dartmouth.edu
@author Based on com.mortbay.HTTP.*.ProxyHandler

Constructors

public `SfUserAgent`( )
public `SfUserAgent`(java.util.Properties properties)

Usage Constructor from properties. Calls setProperties. Three properties are defined for this handler: certDir, useMacs, and authenticateServer.

certDir is a directory that contains bootstrap certificates, and where new certificates or keys may be stored.

useMacs is a boolean parameter indicating whether the client should try to use the MAC protocol to speed requests.

authenticateServer is a boolean parameter that indicates whether the client should check for a proof of document authenticity from the server.

Parameters
`properties` - Configuration properties

Methods

public History `getHistory`( )
public IncomingResponse `getHTTP`(java.net.InetAddress inetAddress, int port, java.net.URL url, com.mortbay.HTTP.HttpRequest request)

Usage If the get fails, the error comes out as a PageException, which the previous version of getHTTP goes ahead and squirts back to the browser. This method is factored out so it can be called in other contexts other than the proxy, such as by the experimental testing harness `timingexp.HttpExp`.

public static byte `getRequestAsBytes`(com.mortbay.HTTP.HttpRequest request)

Usage Translate a Jetty HttpRequest into a bytestring for hashing. This method is used by servlet.PSHandler, too. Yuk; it should be factored into a Tools class or somewhere more reasonable.

public void `handle`(com.mortbay.HTTP.HttpRequest request, com.mortbay.HTTP.HttpResponse response)

Usage Handle proxy requests. Jetty sends requests coming in from the browser to this method.

Parameters

- `request` - the request from the browser
- `response` - the object that collects the response to return to the browser. It is returned once handle() returns.

public void `setProperties`(java.util.Properties properties)

Usage Configure from properties. This handler doesn't support dynamic reconfiguration.

Parameters

- `properties` - configuration.

Class **StateRef**

public class StateRef
extends java.lang.Object

A class that lets a servlet incrementally build a response by tweaking the parameters that appeared in a request.

Class **Tool**

public class Tool
extends java.lang.Object

A rudimentary manual tool for setting up and packaging keys and delegations. It was originally called from the command line while I experimented with different delegations and keys; now its main methods are called from `PrincipalManager`.

CONSTRUCTORS

public `Tool`( )

METHODS

public static SignedCertificate `generateAuthCertificate`(sdsi.SDSIPublicKey issuerPublic, sdsi.SDSIPrivateKey issuerPrivate, sdsi.Subject subject, java.lang.String restrictionTag, boolean propagate, int validDays)

Usage The central work of creating a delegation; used by `PrincipalManager`.

public static void `generateAuthCertificate`(java.lang.String destFilename, java.lang.String issuerFilename, java.lang.String subjectFilename, java.lang.String restrictionTag, boolean propagate, int validDays)

Usage Standalone delegation generator.

public static SignedCertificate `generateDefCertificate`(sdsi.SDSIPublicKey issuerPublic, sdsi.SDSIPrivateKey issuerPrivate, java.lang.String name, sdsi.Subject subject, int validDays)

Usage The central work of creating a name-binding delegation certificate (what SPKI calls a "def").

public static void `generateDefCertificate`(java.lang.String destFilename, java.lang.String issuerFilename, java.lang.String name, java.lang.String subjectFilename, int validDays)

Usage Standalone name-binding delegation generator.

public static SDSIKeyPair `generateKeyPair`( )

Usage The central work of key-pair generation, factored into a component useful to the `PrincipalManager`.

public static void `generateKeyPair`(java.lang.String baseFilename)

Usage Standalone key generation. This code to generate and save an RSA key pair is essentially lifted from sdsi.control.SDSIMainFrame.

public static void `main`(java.lang.String []args)

Usage The original command-line interface to `jp.Tool`.

Class **TweakedServletRequest**

public class TweakedServletRequest **extends** jp.ForwardedHttpServletRequest

Implements the `javax.servlet.http.HttpServletRequest` interface to allow a ServerRef to build a "replacement" request based on an original request, but with changes overlaid. This implementation lets you change the HTTP parameters (name/value pairs that appear separated by '&' characters in a GET request, for example). Based on `ForwardedHttpServletRequest` (and hence `ForwardedServletRequest`), this class overlays the caller's changes encoded in a StateRef over the original request.

@author Jon Howell jonh@cs.dartmouth.edu

Methods

public String `getParameter`(java.lang.String name)

Usage Overrides getParameter to get a parameter from the `ref` tweaks supplied in the constructor.

public Enumeration `getParameterNames`( )

Usage Overrides the default method to get parameter names from the `ref` tweaks supplied in the constructor.

public String `getParameterValues`(java.lang.String name)

Exceptions

Interface **DigestStreamException**

public class DigestStreamException **extends** java.io.IOException

This exception is thrown when a stream does not exhibit the hash it was supposed to have. It is an IOException so that it appears at close() time.

CONSTRUCTORS

public `DigestStreamException`( )
public `DigestStreamException`(java.lang.String p0)

Interface **PageException**

public class PageException **extends** java.lang.Exception

A PageException is a handy way to shape control flow in a server. The server builds an output page, but when it needs to throw an exception that should be reported to the server, it simply throws a PageException (or an appropriate subclass). The PageException itself carries info about the error page to be displayed, and can be easily used in a generic `catch` {} block to present the error to the user.

This is a nicer organization than trying to build fancy error pages in-line where the errors are discovered. They are, after all, exceptions.

FIELDS

public static final int `GATEWAY`

- Shorthand for `HttpServletResponse.SC_BAD_GATEWAY`.

CONSTRUCTORS

public `PageException`(java.lang.Exception source)

Usage Convert another exception into a PageException that knows how to display itself.

public `PageException`(int code, java.lang.Exception source)

Usage Convert an exception into a PageException, supplying the numeric response code to associate with the exception report.

public `PageException`(int code, java.lang.String description)

Usage Create a PageException, supplying both the description and the numeric response code.

public `PageException`(java.lang.String description)

Usage Create an exception with the given description and an SC_INTERNAL_SERVER_ERROR response code.

METHODS

public void `sendResponseTo`(javax.servlet.http.HttpServletResponse resp)

Usage Display the exception as an HTML page.

Parameters

`send` - the HTML page as this response.

public String `toString`( )

Usage Return the exception as a vanilla text string.

Interface **ParseException**

public class ParseException **extends** java.lang.Exception

An Exception thrown by an IncomingResponse object when the incoming stream cannot be parsed as a valid HTTP response.

CONSTRUCTORS

public `ParseException`(java.lang.String str)

Package mail

The `mail` package is an email tool based on Snowflake `sf.Namespace`s. It has a graphical interface, and exploits the Snowflake user-centric model of naming and distribution.

Classes

Class **CategoryView**

public class CategoryView **extends** java.rmi.server.UnicastRemoteObject **implements** sf.Namespace, sf.Program

A namespace that abstracts another namespace by binding each Message object in the other namespace to the category it belongs to.

@classConcise true
@todo implementation incomplete.

CONSTRUCTORS
public `CategoryView`( )

Class **HeaderView**

public class HeaderView **extends** java.rmi.server.UnicastRemoteObject **implements** sf.Namespace

View a collection of messages according to their values for a given header. The email application installs one of these Namespace objects to present the user with (for example) a by-Subject or by-From view of his email box.

@classConcise true

CONSTRUCTORS
public `HeaderView`(mail.Message target)

Class **Mailbox**

public class Mailbox **extends** java.rmi.server.UnicastRemoteObject **implements** sf.Container, sf.Program, java.io.Serializable

A Mailbox is an `sf.Container` that holds a collection of mail. It can be abstracted over by other Namespaces to merge mailboxes or filter or sort them by different properties.

@classConcise true

CONSTRUCTORS
public `Mailbox`( )
public `Mailbox`(java.io.InputStream is)

Class **MailPanel**

public class MailPanel **extends** java.awt.Panel

An awt GUI view of a Mailbox, which is a collection of Messages in a Namespace bound to names that represent some property of each message.

CONSTRUCTORS

public `MailPanel`(sf.Namespace rtparam, java.lang.String nspathparam, gb.Browser brparam)

METHODS

public Insets `getInsets`( )

Class **MailPointer**

public class MailPointer **extends** java.lang.Object

Bindings between messages and “category” objects that reflect how the user has sorted each message.

Class **Message**

public class Message **extends** java.rmi.server.UnicastRemoteObject **implements** sf.Namespace, java.io.Serializable

A single immutable message, preserved as it arrived from the mail system.

@classConcise true

CONSTRUCTORS

public `Message`(java.io.BufferedReader rdr)

METHODS

public String `getHeader`(java.lang.String header)

Usage Return the value of the specified header.

Parameters

`header` - omit the ':'. Example: String messageId = getHeader('Message-Id');

Class **MSU**

public class MSU **extends** java.rmi.server.UnicastRemoteObject **implements** sf.Namespace, java.io.Serializable

Message Storage Unit includes the read-only Message object (the original thing received over the network), plus the user's local annotations, which are mutable.

@todo This is really a useless class. Wouldn't a container with a (Message) message and a (Container) annotations in it do just as well?
@classConcise true

CONSTRUCTORS

protected `MSU`( )
public `MSU`(java.io.BufferedReader br)
public `MSU`(mail.Message m)

METHODS

public HashNS `getAnnotations`( )

Usage Return a namespace which lists the user's annotations on this message.

public Message `getMessage`( )

Usage Return the Message object this unit represents.

Class **MSUPanel**

public class MSUPanel **extends** java.awt.Panel

Display a message in an awt window. A GUI view of an MSU (and the message it contains).

CONSTRUCTORS

public `MSUPanel`(sf.Namespace rtparam, java.lang.String nspathparam, gb.Browser brparam)

METHODS

public Insets `getInsets`( )

Class **SubjectView**

public class SubjectView **extends** java.rmi.server.UnicastRemoteObject **implements** sf.Namespace, sf.Program

A namespace that abstracts another namespace by binding each Message object in the other namespace to its subject line.

@classSummaryOnly true
@deprecated A specific version of what is now `HeaderView`.

Package proof

This package implements the proof verification (server tools) and proof construction (client tools) components of Snowflake sharing and security.

The classes in this package are sorted into three categories. The first category are those classes that manage proof verification, typically what a server might do:

- Proof
- HashProof
- InvalidProofException
- MacProof
- NameLeftMonotonicity
- QuotingRule
- SignedCertificateProof
- TrivialProof
- TwoStepProof

The second category include other tools, notably the client's Prover tool:

- KeyTools
- ProofCache
- Prover2
- SDSIKeyPair

The third category are *deprecated* parts of a previous version of the Prover tool, and have been omitted:

- Prover
- AuthClosure
- BasicUnlockKey
- HashEquivalence
- UseAuth
- NameNode
- DefNameNode
- HashNameNode
- RootNameNode

Classes

Class **HashProof**

public class HashProof
extends proof.Proof

This proof proves that a principal speaks for itself; but it considers the fact that the principal has multiple representations (original plus hashes). That is, the hash of a principal is just an unambiguous shorthand notation for the principal itself. In the logic, the hash is formally treated as a separate principal (hence the need for this proof class), but we also assume that

$$H_A = A$$

That is,

$$(H_A \Rightarrow A) \wedge (A \Rightarrow H_A)$$

Constructors
public `HashProof`(sdsi.SDSIPrincipal thePrincipal, boolean hashIsSubject, java.lang.String hashType)
public `HashProof`(sdsi.sexp.SexpList list)

Methods
protected void `directVerify`( )

Usage verify that the proof steps are indeed valid, and that they combine as advertised to show the claimed result

public SDSIPrincipal `getIssuer`( )
public Subject `getSubject`( )
public Tag `getTag`( )

Class **KeyTools**

public class KeyTools
extends java.lang.Object

A batch of tools related to parsing S-expressions from files and streams, and parsing from S-expressions `Proof`s, `SDSIKeyPair`s, and other Snowflake extensions to Morcos' `sdsi` package.

Constructors
public `KeyTools`( )

Methods
public static boolean `arePrincipalsEquivalent`(sdsi.SDSIObject subjectObj, sdsi.SDSIObject issuerObj)

Usage Check for equivalence up to hash. Can't tell if two different hashes are equivalent, of course, but can help when one is a principal and the other is its hash.

public static SDSIPrivateKey `getPrivateKey`(java.lang.String filename)

Usage Parse a private key out of a Unix file

public static SDSIPublicKey `getPublicKey`(java.lang.String filename)

Usage Parse a public key out of a Unix file

public static ObjectHash `hashObject`(byte []object)

Usage Hash a bytestream and return a SPKI "ObjectHash," a principal that identifies that particular bag of bytes.

Parameters
`object` - the array of bytes to hash.

public static ObjectHash `hashStream`(java.io.InputStream inStream)

Usage Hash a bytestream and return a SPKI "ObjectHash," a principal that identifies that particular bag of bytes.

Parameters
`inStream` - the stream of bytes to hash.

public static SDSIObject `parseBytes`(byte []buf)

Usage Parse a general SDSIObject (including Snowflake extensions) out of a byte buffer.

public static SDSIObject `parseString`(java.lang.String str)

Usage Parse a general SDSIObject (including Snowflake extensions) out of a String.

public static SDSIObject `processFile`(java.io.File file)

Usage Parse a general SDSIObject (including Snowflake extensions) out of a Unix file.

Parameters
`file` - Java File object pointing at the file.

public static SDSIObject `processFilename`(java.lang.String filename)

Usage Parse a general SDSIObject (including Snowflake extensions) out of a Unix file.

Parameters
`filename` - String path name of the file.

Class **MacProof**

public class MacProof **extends** proof.Proof

An object speaks for the hash of a (secret) MAC (Message Authentication Code, which term I'm probably using incorrectly) if we can present a hash of strcat(the object, the secret MAC). That would mean that the holder of the secret MAC allowed its hash to be taken with the object; it's essentially how one "signs" an object using a secret number.

This proof shows that an ObjectHash speaks for another ObjectHash, where the first is the hash of (object,MAC) and the second is hash of just the MAC. For .verify() to succeed, this proof needs to be supplied in advance with a pointer to the object to hash as well as the secret MAC. (These pointers are obviously not transmitted with the object over the network.)

To verify proofs that depend on an instance of this class, a server must supply in advance the appropriate MAC binding that it accepts. That is, MAC signed requests are not self-evident like a public key; they depend on a prior agreement between client and server. The `prepareVerify` method is how the server indicates to the verify method its understanding of that prior agreement.

This proof includes belief in a particular application of the handoff rule...
@todo this is one place where we'd decide how often we believed in it.

CONSTRUCTORS

public `MacProof`(byte []object, byte []mac)

public `MacProof`(byte []object, sdsi.ObjectHash objectHash, byte []mac, sdsi.ObjectHash macHash)

Usage Use this constructor if you already know the MAC's hash (saves extra hash computations)

public `MacProof`(sdsi.sexp.SexpList list)

Usage Parse an S-expression into a MacProof object. In a sense, attach the local verify methods to the remotely-supplied data (proof text).

METHODS

protected void `directVerify`( )

Usage verify that the proof steps are indeed valid, and that they combine as advertised to show the claimed result

public SDSIPrincipal `getIssuer`( )

public Subject `getSubject`( )

public Tag `getTag`( )

Usage This definition of MAC'ing doesn't allow for any tag expression. One could imagine a form that did.

public void `prepareVerify`(byte []object, byte []mac)

Usage Tell me the object and secret mac that the hash corresponds to, so that verify() will work when called in the context of the rest of the proof. If what you tell me doesn't convince me, that's fine; we'll just let verify() fail later.

Class **NameLeftMonotonicity**

public class NameLeftMonotonicity **extends** proof.Proof

Proof of a conclusion that depends upon the left-monotonicity property of names, Axiom E17.

CONSTRUCTORS

public `NameLeftMonotonicity`(proof.Proof p0, sdsi.sexp.SexpString []suffixNames)

Usage Construct a new proof from a premise ($B \Rightarrow A$) and the string of suffixes to append to both principals. Notice that you can supply a string of name suffixes, so that this single proof step collapses a series of n applications of Axiom E17.

public `NameLeftMonotonicity`(sdsi.sexp.SexpList list)

Usage Parse the proof out of an S-Expression.

METHODS
protected SDSIPrincipal `concatenateName`(sdsi.SDSIPrincipal p)
protected void `directVerify`( )

Usage verify that the proof steps are indeed valid, and that they combine as advertised to show the claimed result

public Proof `getChildProof`(int i)
public SDSIPrincipal `getIssuer`( )
public Subject `getSubject`( )
public Tag `getTag`( )
protected void `setupIssuerSubject`( )
public Proof `substituteProof`(int i, proof.Proof subProof)

Class **Proof**

public abstract class Proof **extends** sdsi.SDSIObject

The abstract superclass of the "self-verifying" proofs. When a server receives a proof from a client, it arrives as a SPKI (`sdsi`) S-expression that gets parsed into a Proof class. The class file is loaded locally (so that the client cannot fool the server by sending a proof with a `verify() { return true; }` method).

The server can ask a proof if its conclusion is valid, or if the proof is valid *and* it supports a proposed statement.

CONSTRUCTORS
public `Proof`( )

METHODS
protected abstract void `directVerify`( )

Usage Verify that the proof steps are indeed valid, and that they combine as advertised to show the claimed result. If the method returns without throwing InvalidProofException, the proof was valid. Subclasses should implement this method to verify the statement they represent.

public Proof `getChildProof`(int i)

Usage If this proof contains an ith subproof, return it. (i>=0)

public abstract SDSIPrincipal `getIssuer`( )

Usage who this proof ultimately speaks for public abstract Subject `getSubject`( )

Usage the subject is the principal who stands to gain from this proof, for the proof shows that he speaks for the issuer (possibly with restrictions).

public abstract Tag `getTag`( )

Usage the tag represents the set of requests this proof is valid for (SPKIwise, this is the output of AIntersect.)

public static Proof `parse`(sdsi.sexp.SexpList l)

Usage Parse the given SexpList into a Proof object.

Exceptions

`sdsi.sexp.SexpParseException` - if `l` does not represent a Proof we understand.

public List `preorderCertificates`( )

Usage Return a list of the certificates involved in this proof in preorder-traversal order. Used in proof digestion.

public List `preorderIssuers`( )

Usage Return a list of the issuers involved in this proof in preorder-traversal order. Used in proof digestion.

public List `preorderProofs`( )

Usage Return a list of the subproofs (lemmas) involved in this proof in preorder-traversal order. Used in proof digestion.

public Proof `substituteProof`(int i, proof.Proof subproof)

Usage Substitute the *i*th subproof of this proof with the supplied one, returning a new copy of myself (don't change me). The idea is that we're substituting identical lemmas with different internal state (already-verified objects representing the same statement). If we do this substitution after verifying this object, then we should either clear our own verified flag, or ensure at substitution time that the new proof's conclusion is the same as the one we're substituting out.

public void `verify`( )

Usage Verify that the conclusion this object claims is valid in the logic of restricted delegation.

public void `verify`(sdsi.SDSIPrincipal issuer, sdsi.Subject subject, sdsi.Tag tag)

Usage Verify that the proof is valid, and that it shows that the parameter subject speaks for the parameter issuer regarding the parameter tag.

Class **ProofCache**

public class ProofCache
extends java.lang.Object

An cache of proofs on the server side of a connection. It can replace new proofs with older proofs that are identical in SDSI representation but which have interesting transient data, such as the bit that indicates that we already verified the proof. Useful for servers caching and verifying proofs from clients.

CONSTRUCTORS

public `ProofCache`( )

METHODS

public int `size`( )
public Proof `substitute`(proof.Proof parent)

Class **Prover2**

public class Prover2
extends java.lang.Object

This class is a utility for programs acting as Snowflake clients. It manages a collection of delegations, including authority over certain principals. This tool is described in the dissertation in Section 9.4.

When I say "principal," I mean it as in Snowflake (just about anything even remotely principal-like), not as in SDSI, where only SDSIPrincipals qualify.

FIELDS

public boolean `saveCreatedProofs`

- A flag to turn off when doing a certain performance evaluation, the RMI/ssh experiment, where I want to know how long it takes to create the authorization.

public IndentWriter `iw`

- used for debugging proof lookup

CONSTRUCTORS

public `Prover2`(java.lang.String dirname)

Usage Create a new Prover2 tool.

METHODS

public Proof `createAuth`(sdsi.Subject subject, sdsi.SDSIObject issuer)

Usage Create a delegation that shows that subject ⇒ issuer.

public Proof `createAuth`(sdsi.Subject subject, sdsi.SDSIObject issuer, sdsi.SDSIPublicKey publicKey)

Usage Create a delegation that shows that subject ⇒ issuer.

Parameters

`publicKey` - `subject` is in fact this public key.

public Proof `createAuth`(sdsi.Subject subject, sdsi.SDSIObject issuer, sdsi.Tag tag, sdsi.SDSIPublicKey publicKey)

Usage Create a delegation that shows that subject $\stackrel{\text{tag}}{\Rightarrow}$ issuer.

public void `digestProof`(proof.Proof p)

Usage When someone sends us a proof, this method takes it all apart and saves all the certificates. We can use the digested parts later to build our own proof.

public void `dumpProofs`( )

Usage Dump the proofs cached in the prover. A debugging method.

public Set `getFinalPrincipals`( )

Usage Get the set of principals we consider “final:” those public keys for which we control the corresponding private key, for example, or any other principal that we can cause to say something.

public Hash `getIdentityHash`( )

Usage Get a hash abbreviation for my identity.

public SDSIPrivateKey `getIdentityPrivateKey`( )

Usage Return the private key corresponding to my identity, if my identity is a public key.

public SDSIPublicKey `getIdentityPublicKey`( )

Usage Get some unique notion of identity, by which the caller means he hopes there aren’t multiple public keys I control.

public String `getName`(java.lang.Object obj)

Usage Get a name, secure or mnemonic, for the object. Used for debugging, since it can help you tell keys apart more easily than you might with a hex dump. Not useful for production use, since mnemonic names are easily faked. That’s the point of secure names.

public List `getNames`(sdsi.SDSIPrincipal subject, int numDesired)

Usage Finds every possible name for `p` rooted in a public key for which we have the private key. Algorithm is BFS, so that we can stop once we find a few good, short names.

Parameters

`numDesired` - the maximum number of name chains to return. Specify -1 to completely explore the name graph.

public Set `getPrincipals`( )

Usage Get the entire set of principals currently known to this Prover.

public Set `getPrincipalsByType`(java.lang.Class c)

Usage Get the set of principals that belong to class `c`.

public SDSIPrivateKey `getPrivateKeyForPublic`(sdsi.SDSIPublicKey publicKey)

Usage Map a public key to a private key.

public Proof `getProof`(sdsi.SDSIPrincipal issuer, sdsi.Subject subject, sdsi.Tag authTag)

See Also

- `proof.Prover2.getProofString(SDSIPrincipal,Subject,Tag)`

public String `getProofName`(proof.Proof proof, boolean longForm)

Usage Produce a nice string representation for a proof's conclusion.

Parameters

`longForm` - if true, principals are followed by the class that defines them, and the restriction tag is printed.

public Set `getProofs`( )

Usage Get the entire set of proofs currently known to this Prover.

public String `getProofString`(sdsi.SDSIPrincipal issuer, sdsi.Subject subject, sdsi.Tag authTag)

Usage find a proof that the request speaks for the issuer (ultimate server) regarding all of the statements in authTag.

Algorithm is a BFS over the graph of proofs.

When we find a principal for which we hold the corresponding private key, we're done. (We could find any principal that we could "make" equivalent to the subject, but short of creating a new certificate to do so, which takes a private key, the only current alternative would be to find the exact request itself.)

public SDSIPublicKey `getPublicKeyForHash`(sdsi.Hash hash)

Usage Map a hash back to a public key.

public SDSIPublicKey `getPublicKeyForPrincipal`(sdsi.SDSIObject obj)

Usage Map a public key, hash, or SDSIKeyPair to a public key object.

public SDSIPublicKey `getPublicKeyForPrivate`(sdsi.SDSIPrivateKey privateKey)

Usage Map a private key to the corresponding public key.

public String `getSecureName`(java.lang.Object obj)

Usage Get a secure name for the object. That's a name defined with SPKI name bindings relative to a principal we consider final. A final principal might be one we control, like a public key for which we have the corresponding private key.

public String `getShortClassName`(java.lang.Object o)

Usage Used by getName().

public SDSIObject `introduceObject`(sdsi.SDSIObject so)

Usage Introduce an SDISObject to this Prover. If it's a proof or delegation, it will get cached and used when a proof is requested later.

public SDSIObject `introduceObject`(sdsi.SDSIObject so, boolean persist)

Usage Introduce an SDISObject to this Prover. If it's a proof or delegation, it will get cached and used when a proof is requested later.

Parameters

`persist` - If true, the object will also be saved in the cache dir specified in the constructor.

public void `introducePrincipal`(sdsi.SDSIObject so)

Usage Introduce a principal. Useful, among other times, for introducing a public key when the Prover might encounter proofs that supply only hashes of the key.

public boolean `isFinal`(sdsi.SDSIObject s)

Usage Return true if the subject is one for which we control the private key here.

public void `loadCache`( )

Usage Pulls in any files in the directory `dirname` that have changed since we last checked the directory.

public static void `main`(java.lang.String []args)

Usage A test/debug function. Tries to name all of the objects loaded into the cache.

public Proof `makeProof`(sdsi.SDSIPrincipal issuer, sdsi.Subject subject, sdsi.Tag authTag)

Usage Calls getProof(), but if proof doesn't exist, will look for a proof for an issuer we control, and sign off on a delegation for the last step subject ⇒ myIssuer.

public static String `staticGetName`(java.lang.Object obj)

Usage Get a debugging name for an object, even if you have no Prover available.

public void `stats`( )

Usage Print out some simple stats on the objects this Prover has collected.

Class **QuotingRule**

public class QuotingRule
extends proof.Proof

CONSTRUCTORS

public `QuotingRule`(sdsi.Quoting issuer, sdsi.Subject subject)
public `QuotingRule`(sdsi.sexp.SexpList list)

METHODS

protected void `directVerify`( )

Usage verify that the proof steps are indeed valid, and that they combine as advertised to show the claimed result

public SDSIPrincipal `getIssuer`( )
public Subject `getSubject`( )
public Tag `getTag`( )

Class **SDSIKeyPair**

public class SDSIKeyPair
extends sdsi.SDSIObject

An S-expression that holds both a private and a public key in the same file, so it's really easy to tell that they go together. This is a convenient way to package private keys so that we don't lose track of the public key that it goes with.

CONSTRUCTORS
public `SDSIKeyPair`(sdsi.SDSIPrivateKey privateKey, sdsi.SDSIPublicKey publicKey)
public `SDSIKeyPair`(sdsi.sexp.SexpList l)

Usage Parse the given SexpList into a SDSIKeyPair object.

METHODS
public SDSIPrivateKey `getPrivateKey`( )
public SDSIPublicKey `getPublicKey`( )

Class **SignedCertificateProof**

public class SignedCertificateProof
extends proof.Proof

This proof verifies a "self-evident statement" of the public key signature variety. That is, it verifies

$$A \ \textbf{says} \ B \overset{T}{\Rightarrow} A$$

when A is a public key and we have A's signature on an S-expression that says $B \overset{T}{\Rightarrow} A$.

@todo This proof includes belief in an application of the handoff rule. This is one place where we'd decide how often we believed in it.

CONSTRUCTORS
public `SignedCertificateProof`(sdsi.sexp.SexpList list)
public `SignedCertificateProof`(sdsi.SignedCertificate sc)
public `SignedCertificateProof`(sdsi.SignedCertificate sc, proof.Proof handoff)

METHODS
protected void `directVerify`( )

Usage verify that the proof steps are indeed valid, and that they combine as advertised to show the claimed result

public Proof `getChildProof`(int i)
public SDSIPrincipal `getIssuer`( )
public Subject `getSubject`( )
public Tag `getTag`( )
public Proof `substituteProof`(int i, proof.Proof subProof)

Class **TrivialProof**

public class TrivialProof **extends** proof.Proof

This proof proves that a principal speaks for itself ($A \Rightarrow A$). It's kind of silly to reify this as an explicit object, but it avoids putting potentially-confusing special-case code in the proof verifier

Constructors

public `TrivialProof`(sdsi.SDSIPrincipal thePrincipal)
public `TrivialProof`(sdsi.sexp.SexpList list)

Methods

protected void `directVerify`( )
Usage verify that the proof steps are indeed valid, and that they combine as advertised to show the claimed result

public SDSIPrincipal `getIssuer`( )
public Subject `getSubject`( )
public Tag `getTag`( )

Class **TwoStepProof**

public class TwoStepProof **extends** proof.Proof

This proof verifies a proof involving restricted transitive delegation, Theorem E6, that subsumes Axiom E1 as well.

Constructors

public `TwoStepProof`(proof.Proof p0, proof.Proof p1)
Usage Construct a two-step proof from two appropriate lemmas.

public `TwoStepProof`(sdsi.sexp.SexpList list)
Usage Parse a proof from an S-Expression.

Methods

protected void `directVerify`( )
Usage verify that the proof steps are indeed valid, and that they combine as advertised to show the claimed result

public Proof `getChildProof`(int i)
public SDSIPrincipal `getIssuer`( )
public Subject `getSubject`( )
public Tag `getTag`( )
public Proof `substituteProof`(int i, proof.Proof subProof)

Exceptions

Interface **InvalidProofException**

public class InvalidProofException
extends java.lang.Exception

The exception thrown by `Proof.verify` if the proof is not valid.

CONSTRUCTORS

public `InvalidProofException`(java.lang.String s)

Package relational

This package implements a relational database. It is stored in-core, so persistence must be supplied with some external mechanism; may I suggest `Icee`?

The database is very tightly bound to Java types. It started as a simple way to index "back pointers" rather than storing lists of back pointers explicitly in objects. The more I added relational-like features to it, though, the more I realized that relational semantics are enough significantly different than object semantics that the two do not blend as well as one would hope.

The database supports indexing for fast lookups.

Interfaces

Interface **Database**

```
public interface Database
extends java.rmi.Remote
```

A database can do select-like operations. Relational objects need to work with the database to get created, so the database can track them.

METHODS

public void `createIndex`(relational.FieldDescriptor fd)

Usage Hint to the database the fields you want indexed

public ResultSet `evaluateSelect`(relational.Select s)

Usage Every database can perform the select() operation. It's neat-o because it can "invert" pointers.

public void `insert`(relational.Relational ro)

Usage All Relational objects in ros[] should be of the same class. ros.length should be greater than 0. (duh)

public void `insert`(relational.Relational []ros)
public void `noop`( )

Usage Do nothing. Verifies that the database server is accessible.

public void `shutdown`( )
public void `update`(relational.Relational ro)
public void `update`(relational.Relational []ros)

Interface **OrderBy**

```
public interface OrderBy
extends java.io.Serializable
```

A clause to attach to a select statement to request ordering.

METHODS

public ResultSet `order`(relational.ResultSet rs)

Usage Database calls this to sort the ResultSet rs before returning it to the caller.

Interface **ResultSet**

```
public interface ResultSet
extends java.io.Serializable
```

A ResultSet is an evaluated query or a table – a static list of rows.

METHODS

public ColumnSpec `getColumnSpec`( )
public Enumeration `getEnumeration`( )
public FromClause `getFromClause`( )

public Vector `getVector`( )
public boolean `hasMember`(relational.Row o)
public Iterator `iterator`( )
public int `size`( )

Interface **Row**

public interface Row
extends java.io.Serializable

A Row packages up objects that form a row of results; it can be accessed by a ColumnSpec requesting specific columns.

METHODS
public ColumnSpec `getColumnSpec`( )
public Object `getField`(relational.FieldDescriptor fd)
public Object `getField`(int tableIndex, relational.FieldDescriptor fd)
public FromClause `getFromClause`( )
public Relational `getTable`(java.lang.Class c)
public Relational `getTable`(relational.FromClause fc, int table)
public Relational `getTable`(java.lang.String tableName)
public boolean `supports`(relational.ColumnSpec cs)

Classes

Class **BasicRow**

public class BasicRow
extends java.lang.Object
implements Row

A simple (inefficient for network transfer) implementation of Row – it stores every object that has a field referenced by the columnspec.

(A more efficiently serializable implementation would store only the necessary fields / primary keys.)

CONSTRUCTORS
public `BasicRow`(relational.FromClause fromClause, relational.Relational []data)
public `BasicRow`(relational.FromClause fromClause, relational.Row row, int index, relational.Relational oneMore)

METHODS
public boolean `equals`(java.lang.Object o)
public ColumnSpec `getColumnSpec`( )
public Object `getField`(relational.FieldDescriptor fd)
public Object `getField`(int tableIndex, relational.FieldDescriptor fd)
public FromClause `getFromClause`( )
public Relational `getTable`(java.lang.Class c)

public Relational `getTable`(relational.FromClause fc, int table)
public Relational `getTable`(java.lang.String tableName)
public int `hashCode`( )
public boolean `supports`(relational.ColumnSpec cs)

Class **CheckDatabase**

public class CheckDatabase
extends java.lang.Object

See if an RMI Database is accessible by trying to invoke its noop method. Written when I was trying to get SSL to work.

CONSTRUCTORS
public `CheckDatabase`( )

METHODS
public static void `main`(java.lang.String []argv)

Class **ClumpRelational**

public abstract class ClumpRelational
extends relational.Relational

ClumpRelational Objects in this class have primary keys that encode their database membership, saving four bytes per class member.

CONSTRUCTORS
public `ClumpRelational`(relational.Database db)

METHODS
public Database `getDatabase`( )
protected static void `growMap`(int pastHere)

Class **ColumnSpec**

public class ColumnSpec
extends java.lang.Object
implements java.io.Serializable

A ColumnSpec is an ordered list of unique columns of a FromClause.

CONSTRUCTORS
public `ColumnSpec`( )

METHODS
public static ColumnSpec `create`(relational.FromClause fromClause, int []indices, relational.FieldDescriptor []fields)
public static ColumnSpec `create`(relational.FromClause fromClause, java.lang.String []names, relational.FieldDescriptor []fields)
public int `findField`(relational.FieldDescriptor fieldIdent)
public Class `getDeclaringClass`(int field)
public FieldDescriptor `getField`(int field)
public Object `getField`(relational.Row source, int field)

public FromClause `getFromClause`( )
public int `getNumFields`( )
public int `getTableIndex`(int field)
public String `getTableName`(int field)
public Class `getType`(int field)
public boolean `supports`(relational.FieldDescriptor fieldIdent)
public String `toString`( )

Class **DirectRelational**

public abstract class DirectRelational
extends relational.Relational

The Database that serves a DirectRelational object (row) is stored as a reference in a field along with each such object.

CONSTRUCTORS
public `DirectRelational`(relational.Database db)

METHODS
public Database `getDatabase`( )
public void `setDatabase`(relational.Database db)

Class **FieldDescriptor**

public abstract class FieldDescriptor
extends java.lang.Object
implements java.io.Serializable

An object that identifies a field of a "table" (class). These come in a few varieties.

CONSTRUCTORS
public `FieldDescriptor`( )

METHODS
public static FieldDescriptor `get`(java.lang.Class c)

Usage A field that refers to the object defining the row

public static FieldDescriptor `get`(java.lang.reflect.Field f)
public static FieldDescriptor `get`(java.lang.reflect.Field f, java.lang.Class c)

Usage Field descriptor for field f in class c (even if f is a member of a superclass of c)

public abstract Object `get`(relational.Row source)
public abstract Class `getDeclaringClass`( )

Usage which table declares this field

public static FieldDescriptor `getPrimaryKey`(java.lang.Class c)
public abstract Class `getType`( )

Usage which table (Class) this field's value belongs to

Class **FieldDescriptorField**

public class FieldDescriptorField **extends** relational.FieldDescriptor

Describes a regular field of a class.

CONSTRUCTORS

public `FieldDescriptorField`( )

METHODS

public boolean `equals`(java.lang.Object o)
public static FieldDescriptor `get`(java.lang.reflect.Field f, java.lang.Class c)
public Object `get`(relational.Row source)
public Class `getDeclaringClass`( )
public Field `getField`( )
public Class `getType`( )
public String `toString`( )

Class **FieldDescriptorForeign**

public class FieldDescriptorForeign **extends** relational.FieldDescriptor

Describes a "foreign field," that is, a reference to another class.

CONSTRUCTORS

public `FieldDescriptorForeign`( )

METHODS

public boolean `equals`(java.lang.Object o)
public static FieldDescriptor `get`(java.lang.reflect.Field f)
public Object `get`(relational.Row source)
public Class `getDeclaringClass`( )
public Class `getType`( )
public String `toString`( )

Class **FieldDescriptorPrimary**

public class FieldDescriptorPrimary **extends** relational.FieldDescriptor

Describes the "primary field" for this class; that is, the reference to this object itself. Used in queries that specify that one object points to another: the first object's foreign field must match the second object's primary field.

CONSTRUCTORS

public `FieldDescriptorPrimary`( )

METHODS

public boolean `equals`(java.lang.Object o)
public static FieldDescriptor `get`(java.lang.Class c)
public Object `get`(relational.Row source)
public Class `getDeclaringClass`( )

public Class `getType`( )
public String `toString`( )

Class **FieldDescriptorReference**

public class FieldDescriptorReference **extends** relational.FieldDescriptor

CONSTRUCTORS
public `FieldDescriptorReference`( )

METHODS
public boolean `equals`(java.lang.Object o)
public static FieldDescriptor `get`(java.lang.Class c)
public Object `get`(relational.Row source)
public Class `getDeclaringClass`( )
public Class `getType`( )
public String `toString`( )

Class **FromClause**

public class FromClause **extends** java.lang.Object **implements** java.io.Serializable

A FromClause identifies an ordered list of tables, possibly by name. Used in a select statement just as FROM is used in SQL.

CONSTRUCTORS
public `FromClause`( )

METHODS
public static FromClause `create`(java.lang.String []names, java.lang.Class []tables)
public static FromClause `create`(java.lang.String name, java.lang.Class table)
public static FromClause `createAnonymous`(java.lang.Class c)
protected void `ensureUniqueNames`( )
public boolean `equals`(java.lang.Object o)
public int `getIndex`(java.lang.Class table)
public int `getIndex`(java.lang.String name)
public String `getName`(java.lang.Class t)
public String `getName`(int i)
public ColumnSpec `getNaturalColumnSpec`( )
public int `getNumTables`( )
public Class `getTable`(int i)
public Relational `getTableFromRow`(relational.Row source, int tableIndex)
public Relational `getTableFromRow`(relational.Row source, java.lang.String name)
public boolean `hasTable`(java.lang.Class table)
public boolean `subsetOf`(relational.FromClause superfc)
public String `toString`( )
public static FromClause `trimOne`(relational.FromClause fc)
public static FromClause `union`(relational.FromClause fca, relational.FromClause fcb)

Class **InternalDatabase**

public class InternalDatabase **extends** java.rmi.server.UnicastRemoteObject **implements** Database

My implementation of the Database interface. Supports select statements and indexing.

@todo An implementation of Relational needs a way to always be able to invert any pointer. One really crummy mechanism is to keep track of all Relationals of each type, and when asking for the pointers from a given type, iterate through the list of existing guys.
@todo Some small issues with references and never garbage collecting are sure to show up.
@todo subclasses of Relational classes don't work yet.

Constructors

public `InternalDatabase`( )
public `InternalDatabase`(ssh.SSHContext context, sdsi.SDSIPrincipal serverIssuer)

Usage Create an InternalDatabase object that is accessed via RMI-over-SSH.

Parameters
`context` - The SSHContext object to use with the SSH connection.

public `InternalDatabase`(COM.claymoresystems.ptls.SSLContext context)

Usage Create an InternalDatabase object that is accessed via RMI-over-SSL. [I couldn't get RMI-over-SSL working reliably, so I switched to my SSH implementation.]

Parameters
`context` - The SSLContext object to use with the SSL connection.

Methods

public ResultSet `boundAnd`(relational.WhereAnd w, relational.FromClause fc, relational.ResultSet input)
public ResultSet `boundConstant`(relational.WhereConstant wc, relational.FromClause fc, relational.ResultSet input)
public ResultSet `boundIn`(relational.WhereIn win, relational.FromClause fc, relational.ResultSet input)
public ResultSet `boundingSuperset`(relational.Where w, relational.FromClause fc, relational.ResultSet input)

Usage Optimization for Where clauses. Given a Where clause, the FromClause it is scoped over, and some input superset, (quickly) compute a (possibly not-tight) superset of the possible matching rows fitting the FromClause.

The superset can be a loose bound in two ways: First, in the obvious way, it can explicitly list more rows that actually match the request. Second, it can have a weaker type (fromClause). So if fc=*tableA,tableB*, but the superset ResultSet's getFromClause()=*tableA*, then the superset contains the join of its rows with every row in *tableB*, which is a shorthand for a lot of rows.

The latter loose bound is used when computing joins, in fact. One whereClause finds a condition on one table, and expresses it as described above (compactly, listing only the matching rows of tableA). Then the whereJoin() clause can index tableB on the joined column of tableA, filling out the type (fromClause) of the bounding set, and therefore making the bound tighter (because it doesn't end up listing every possible row of tableB with every row of the input superset.)

Often these routines compute an actually-tight superset, at least for the returned fromClause. However, right now the semantics is that whatever results are returned, they are first expanded to the full requested fromClause (by joining in unmentioned tables), then every row of the result is forced through the where expression to verify that it matches.

public ResultSet `boundJoin`(relational.WhereJoin wj, relational.FromClause fc, relational.ResultSet input)
public ResultSet `boundLiteral`(relational.WhereLiteral w, relational.FromClause fc, relational.ResultSet input)
public ResultSet `boundNot`(relational.WhereNot wnot, relational.FromClause fc, relational.ResultSet input)
public ResultSet `boundOr`(relational.WhereOr wor, relational.FromClause fc, relational.ResultSet input)
public void `createIndex`(relational.FieldDescriptor fd)

Usage iterates over all members of the class(es) that declare field, indexing what they point to.

protected Hashtable `createIndex`(relational.FieldDescriptor fd, relational.ResultSet rs)
protected Hashtable `createIndex`(relational.FieldDescriptor fd, relational.ResultSet rs, relational.LikeHash lh)
public ResultSet `evaluateSelect`(relational.Select s)
protected ResultSet `fillByJoin`(relational.ResultSet rsi, relational.FromClause fc)
protected ResultSet `getUniverse`(java.lang.Class c, java.lang.String fromName)
protected ResultSet `getUniverse`(relational.FromClause fc)
public void `indexOneField`(java.util.Hashtable index, relational.Relational r, java.lang.Object target)
public void `indexOneValue`(java.util.Hashtable index, relational.Relational r, java.lang.Object target)
protected void `indexSome`(relational.Relational []ros)
public void `insert`(relational.Relational ro)
public void `insert`(relational.Relational []ros)
protected boolean `isValid`(relational.Row ro)
public void `noop`( )
public ResultSet `primeFromIndexWE`(relational.WhereEquals we, java.lang.String fromName)

public ResultSet `primeFromIndexWL`(relational.WhereLike wl, java.lang.String fromName)
public static Vector `select`(relational.Database db, java.lang.Class fromClass, relational.Where where)

Usage Utility method. Finds all objects of fromClass that match the where clause. This can be used to follow pointers backwards, by using a where clause that checks for a data member that points at the target data item.

public void `shutdown`( )
public void `update`(relational.Relational ro)
public void `update`(relational.Relational []ros)

Class **NumericComparator**

public class NumericComparator **extends** java.lang.Object **implements** java.util.Comparator, java.io.Serializable

A numeric Comparator for sorting.

CONSTRUCTORS
public `NumericComparator`( )

METHODS
public int `compare`(java.lang.Object o1, java.lang.Object o2)

Class **OrderByOne**

public class OrderByOne **extends** java.lang.Object **implements** OrderBy

Sorts results by a single column. (Versus a hypothetical sort that sorts by multiple columns, in priority order.)

CONSTRUCTORS
public `OrderByOne`(relational.FieldDescriptor fd)
public `OrderByOne`(relational.FieldDescriptor fd, java.util.Comparator fieldComp)

METHODS
public static OrderBy `natural`(relational.FieldDescriptor fd)
public ResultSet `order`(relational.ResultSet rs)

Class **Relational**

public abstract class Relational **extends** java.lang.Object **implements** Row

All objects that can be accessed relationally must subclass this abstract class. They all have an “update” method that makes their data “persistent” (although it may be already), and ensures that it is indexed.

Note that this is a “Row.” A Relational instance is a single row from a single table

(Relational class); a compound row made up from multiple tables joined would be some other implementation of Row (like BasicRow).

Serializable Fields
public Object `primaryKey`

Constructors
public `Relational`( )

Methods
public ColumnSpec `getColumnSpec`( )
public abstract Database `getDatabase`( )
public Object `getField`(relational.FieldDescriptor fd)
public Object `getField`(int tableIndex, relational.FieldDescriptor fd)
public FromClause `getFromClause`( )
public Relational `getTable`(java.lang.Class c)
public Relational `getTable`(relational.FromClause fc, int table)
public Relational `getTable`(java.lang.String tableName)
public void `insert`( )
public Object `resolveForeignKey`(java.lang.Class c, java.lang.Object key)
public void `setPrimaryKey`(java.lang.Object pk)
public boolean `supports`(relational.ColumnSpec cs)
public void `update`( )

Class **ResultSetImpl**

public class ResultSetImpl
extends java.lang.Object
implements ResultSet

A list of rows that implement a complete FromClause. If you want rows that perhaps only hold (or report) a couple of specific columns, you need a ResultSetNarrow.

Constructors
public `ResultSetImpl`(relational.FromClause fc)
public `ResultSetImpl`(relational.FromClause fc, java.util.Collection c)
public `ResultSetImpl`(relational.FromClause fc, java.util.Vector v)
public `ResultSetImpl`(relational.ResultSet rs)

Methods
public void `addMember`(java.lang.Object o)
public static ResultSet `cross`(relational.ResultSet ra, relational.ResultSet rb)
public static ResultSet `cross`(relational.ResultSet ra, relational.ResultSet rb, relational.FromClause outputShape)
protected void `fillCache`( )
public ColumnSpec `getColumnSpec`( )
public Enumeration `getEnumeration`( )
public FromClause `getFromClause`( )
public Vector `getVector`( )
protected Object `hashKeyForRow`(relational.Row row)

public boolean **hasMember**(relational.Row o)
public Iterator **iterator**( )
public void **removeMember**(java.lang.Object o)
public int **size**( )

Class **ResultSetNarrow**

public class ResultSetNarrow **extends** java.lang.Object **implements** ResultSet

A ResultSetNarrow holds results shaped into a specific ColumnSpec, not just a FromClause.

Constructors
public **ResultSetNarrow**(relational.ColumnSpec cs)
public **ResultSetNarrow**(relational.ColumnSpec cs, relational.ResultSet rs)
public **ResultSetNarrow**(relational.ResultSet rs)

Methods
public void **addMember**(java.lang.Object o)
protected static ColumnSpec **alignColumnSpec**(relational.ColumnSpec cs, relational.FromClause fc)
protected void **fillCache**( )
public ColumnSpec **getColumnSpec**( )
public Enumeration **getEnumeration**( )
public FromClause **getFromClause**( )
public Vector **getVector**( )
public boolean **hasMember**(relational.Row o)
public boolean **hasSingleton**(java.lang.Object o)
public Iterator **iterator**( )
public void **removeMember**(java.lang.Object o)
public int **size**( )

Class **RMIDatabase**

public class RMIDatabase **extends** java.lang.Object

A Unix command-line interface to instantiate an InternalDatabase and bind it to a well-known name in the localhost's RMIRegistry.

Constructors
public **RMIDatabase**( )

Methods
public static void **main**(java.lang.String []args)

Class **RowTools**

public class RowTools **extends** java.lang.Object

Tools used by classes that implement Row.

CONSTRUCTORS
public `RowTools`( )

METHODS
public static ColumnSpec `getColumnSpec`(java.lang.Object o)

Class **Select**

public class Select
extends java.lang.Object
implements java.io.Serializable

A select statement.

CONSTRUCTORS
public `Select`(java.lang.Class wholeTable, relational.Where whereClause)
public `Select`(relational.ColumnSpec columnSpec, relational.FromClause fromClause, relational.Where whereClause)
public `Select`(relational.ColumnSpec columnSpec, relational.FromClause fromClause, relational.Where whereClause, boolean distinct)
public `Select`(relational.ColumnSpec type, relational.Where whereClause)
public `Select`(relational.FieldDescriptor oneField, relational.Where whereClause)
public `Select`(relational.Select s)

METHODS
public ResultSet `evaluate`(relational.Database db)
public ColumnSpec `getColumnSpec`( )
public FromClause `getFromClause`( )
public Where `getWhere`( )
protected void `init`(relational.ColumnSpec columnSpec, relational.FromClause fromClause, relational.Where whereClause, boolean distinct)
public ResultSet `order`(relational.ResultSet rs)
public void `setDistinct`(boolean distinct)
public void `setOrderBy`(relational.OrderBy ob)
public String `toString`( )

Class **SSHDatabase**

public class SSHDatabase
extends java.lang.Object

Creates a remotely-accessible Database that is accessed using Snowflake-authorized RMI-over-SSH.

CONSTRUCTORS
public `SSHDatabase`( )

METHODS
public static void `main`(java.lang.String []args)
Usage Unix command-line interface.

Class **SSLDatabase**

public class SSLDatabase
extends java.lang.Object

Creates a remotely-accessible Database that is accessed using RMI-over-SSL (PureTLS). (No particular client authorization mechanism yet – it turned out that I could never get SSL over RMI working well.)

CONSTRUCTORS
public `SSLDatabase`( )

METHODS
public static void `main`(java.lang.String []args)

Class **TableDescriptor**

public class TableDescriptor
extends java.lang.Object
implements java.io.Serializable, java.lang.Comparable

Description of a table (class), including a name to identify *which* reference to the table we are talking about. Used in SQL-like select statements that refer to the same table twice, for example, when examining tree or other self-referential structures. Used to build `FromClause`s.

CONSTRUCTORS
public `TableDescriptor`(java.lang.String name, java.lang.Class table)

METHODS
public int `compareTo`(java.lang.Object o)
public boolean `equals`(java.lang.Object o)
public int `hashCode`( )

Class **Where**

public abstract class Where
extends java.lang.Object
implements java.io.Serializable

The abstract Where clause used to specify select statements.

CONSTRUCTORS
public `Where`( )

METHODS
public static Where `always`( )
public static Where `and`(relational.Where w1, relational.Where w2)
public Object `clone`( )
public static Where `equals`(relational.FieldDescriptor fd, java.lang.Object o)
public abstract Object `getChild`(int index)
public abstract int `getChildCount`( )
public String `getShortName`( )
public abstract boolean `includes`(relational.Row ro, relational.Database db)

public String `indentedString`(int level)
public abstract void `setChild`(int index, java.lang.Object child)

Class **WhereAlways**

public class WhereAlways
extends relational.Where

The null Where clause that accepts all rows specified by the FromClause.

@classConcise true

CONSTRUCTORS

public `WhereAlways`( )

Class **WhereAnd**

public class WhereAnd
extends relational.WhereBinary

Conjunction of two other where clauses.

@classConcise true

CONSTRUCTORS

public `WhereAnd`( )
public `WhereAnd`(relational.Where w1, relational.Where w2)

Class **WhereBinary**

public abstract class WhereBinary
extends relational.Where

Abstract superclass for Where clauses with two subclauses (binary operators).

@classConcise true

CONSTRUCTORS

public `WhereBinary`( )
public `WhereBinary`(relational.Where w1, relational.Where w2)

Class **WhereConstant**

public class WhereConstant
extends relational.Where

Used for literal comparisons like (WHERE field = 'foo').

@classConcise true

CONSTRUCTORS

public `WhereConstant`(java.lang.String tableName, relational.Relational constant)

Class **WhereEquals**

public class WhereEquals
extends relational.WhereLiteral

Test for equality between two fields (neither operand is a literal).

@classConcise true

CONSTRUCTORS
public `WhereEquals`(relational.FieldDescriptor fd, java.lang.Object o)

Class **WhereIn**

public class WhereIn **extends** relational.Where

Test for membership of a field in the list of entries (ResultSet) of a subquery. Like SQL's `SELECT ... WHERE X IN (SELECT ...)`

@classConcise true

CONSTRUCTORS
public `WhereIn`( )
public `WhereIn`(relational.FieldDescriptor fd, relational.Select subquery)

Class **WhereJoin**

public class WhereJoin **extends** relational.Where

Join two tables together where two fields are equal.
@classConcise true

CONSTRUCTORS
public `WhereJoin`(relational.FieldDescriptor f1, relational.FieldDescriptor f2)

Class **WhereLike**

public class WhereLike **extends** relational.WhereLiteral

Test where a substring appears in a field.
@classConcise true

CONSTRUCTORS
public `WhereLike`(relational.FieldDescriptor fd, java.lang.String s)
public `WhereLike`(relational.FieldDescriptor fd, java.lang.String s, boolean caseSensitive, boolean startsWith, boolean endsWith)

Class **WhereLiteral**

public abstract class WhereLiteral **extends** relational.Where

Superclass for comparisons that involve a single field and another literal operand.

CONSTRUCTORS
public `WhereLiteral`( )

METHODS
public abstract FieldDescriptor `getFieldDescriptor`( )

Class **WhereNot**

public class WhereNot
extends relational.Where

Invert the sense of a where subclause.

@classConcise true

CONSTRUCTORS

public `WhereNot`( )
public `WhereNot`(relational.Where w1)

Class **WhereOr**

public class WhereOr
extends relational.WhereBinary

Disjunction of two where subclauses.

@classConcise true

CONSTRUCTORS

public `WhereOr`( )
public `WhereOr`(relational.Where w1, relational.Where w2)

Package relational.email

This package implements a relational database schema for an email database. The schema is composed of three primary tables (classes):

- `Message`, an empty row that ties the body and headers together,
- `Body`, a row containing the body text of the document, and
- `Header`, a row containing a single header and its value.

In the future, the schema may be extended to support attachments or the retrieval of "body parts" so that the entire message body does not need to be transferred in one unit.

Properties can be attached to mail with "synthetic headers," which are entries in the header table with a flag set indicating that the header was not in the email as delivered, but added after receipt by an application.

The `Mailbox` class is a tool that can create the schema in a Database and parse Berkeley-style mail folders into it. The `Extract` class can extract messages to an output stream. The `Email` class provides a Swing graphical interface on the email, including drag-and-drop specification of query filters.

Interfaces

Interface **DisplayManager**

public interface DisplayManager

An interface for classes that organize the visual elements of the email application.

METHODS
public void `add`(javax.swing.JComponent c, java.lang.String title)
public FilterView `getFilterView`( )
public MessageView `getMessageView`( )
public void `registerKeyboardAction`(java.awt.event.ActionListener a, java.lang.String s, javax.swing.KeyStroke k, int c)
public void `setVisible`(boolean state)

Classes

Class **Body**

public class Body **extends** relational.ClumpRelational

Part of the Database schema for email. A Body row carries the body text of a message, and connects it to a specific `Message` object.

SERIALIZABLE FIELDS
public String `body`
public Object `msg_fk`

FIELDS
public static FieldDescriptor `f_reference`
public static FieldDescriptor `f_primaryKey`
public static FieldDescriptor `f_body`
public static FieldDescriptor `f_msg`

CONSTRUCTORS
public `Body`(relational.Database db)

METHODS
public Message `getMsg`( )
public void `setMsg`(relational.email.Message m)

Class **ChangeEventMulticaster**

public class ChangeEventMulticaster **extends** java.lang.Object **implements** javax.swing.event.ChangeListener

This class does for senders of ChangeEvent what AWTEventMulticaster does for all the other events. Maintains an immutable linked list of listeners, multicasting any events to all listeners on the list. Bears a striking resemblance to AWTEventMulticaster.
@classConcise true

@author Jon Howell

CONSTRUCTORS
protected `ChangeEventMulticaster`(javax.swing.event.ChangeListener a, javax.swing.event.ChangeListener b)

Class **CommandPanel**

public class CommandPanel
extends javax.swing.JPanel
implements java.awt.event.ActionListener, javax.swing.event.ChangeListener

The user-interface element that captures single keystrokes to enable the user to quickly navigate a panel of commands. Inspired by the fact that I never pull-down menus when I use Pine.

@classConcise true

CONSTRUCTORS
public `CommandPanel`(relational.email.WherePanel wp)

Class **ComposeAction**

public class ComposeAction
extends javax.swing.AbstractAction

Action handler for composing a new message. Sets up the message pane to accept new text, and configures the “send” button.

CONSTRUCTORS
public `ComposeAction`(boolean reply)

METHODS
public void `actionPerformed`(java.awt.event.ActionEvent e)
protected String `getHeader`(java.lang.String name)

Class **DragTableUI**

public class DragTableUI
extends javax.swing.plaf.basic.BasicTableUI

Need to modify the BasicTableUI to not snarf drag events, or else it ends up fighting with the drag-n-drop code.

@author jonh (Jon Howell)

CONSTRUCTORS
public `DragTableUI`( )

METHODS
protected MouseInputListener `createMouseInputListener`( )
public static ComponentUI `createUI`(javax.swing.JComponent c)
public void `dragStarted`( )

Class **EditorComboBox**

public class EditorComboBox
extends javax.swing.JComboBox
implements javax.swing.CellEditor

EditorComboBox: A CellEditor JComboBox subclass for use with Trees (and possibly tables). Swiped from O'Reilly's Java Swing book, ch17.

@classConcise true

CONSTRUCTORS

public `EditorComboBox`(java.lang.Object []list)

Class **EditorTextField**

public class EditorTextField
extends javax.swing.JTextField
implements javax.swing.CellEditor

A CellEditor.JTextField subclass for use with Trees (and possibly tables). Swiped from O'Rielly Java Swing book, ch17.

@classConcise true

CONSTRUCTORS

public `EditorTextField`( )
public `EditorTextField`(int w)
public `EditorTextField`(java.lang.String s)
public `EditorTextField`(java.lang.String s, int w)

Class **Email**

public class Email
extends java.lang.Object

Instantiate from the Unix command line a graphical application that reads and composes email using the relational email database schema.

CONSTRUCTORS

public `Email`( )

METHODS

public static DisplayManager `getDisplayManager`( )
public static void `main`(java.lang.String []args)

Class **Extract**

public class Extract
extends java.lang.Object

Extract email from the relational database and spit it out as a Berkeley-style mail folder. Originally designed for debugging: by parsing big mailboxes into and out of my relational databases, I could run a *diff* to determine if the mail had been munged during the parse.

CONSTRUCTORS

public `Extract`( )

Methods

public static Body `getBody`(relational.Database db, relational.email.Message m)
public static ResultSet `getSortedHeaders`(relational.Database db, relational.email.Message m)
public static ResultSet `getSortedMessages`(relational.Database db, relational.Where whereClause, java.lang.String sortHeader, java.lang.String defaultValue, java.util.Comparator comp)
public static void `main`(java.lang.String []argv)

Class **FilterModel**

public class FilterModel
extends javax.swing.table.AbstractTableModel
implements javax.swing.event.ChangeListener

The model part of an MVC pattern for specifying an email filter query. The output of the query is what appears in the index of the email GUI.

Constructors

public `FilterModel`( )

Methods

public int `getColumnCount`( )
public String `getColumnName`(int modelIndex)
public Database `getDatabase`( )
public Message `getMessageAt`(int row)
public int `getRowCount`( )
public Object `getValueAt`(int row, int col)
protected void `loadData`( )
public void `setDatabase`(relational.Database db)
public void `stateChanged`(javax.swing.event.ChangeEvent e)

Class **FilterView**

public class FilterView
extends javax.swing.JPanel

The view part of the MVC pattern for the query filter control. I guess it's also the controller. Crazy Swing.

Constructors

public `FilterView`( )

Methods

public Database `getDatabase`( )
public FilterModel `getModel`( )
public Message `getSelectedMessage`( )
public WherePanel `getWherePanel`( )
public void `setMessageViewer`(relational.email.MessageView viewer)

Class **Header**

public class Header
extends relational.ClumpRelational

Part of the email database schema. A Header is a single header from a message. Headers with multi-line values are collapsed into a single Header row, so that their value can be easily retrieved. Headers have an order field that specify the order the headers appeared in the received message.

Synthetic headers are properties added to a message after receipt.

SERIALIZABLE FIELDS

public String `name`
- The name of the header; the part before the colon.

public String `whitespace`
- Any whitespace that got removed between the colon and the value. Storing this junk lets us reconstruct the original message precisely.

public String `value`
- The value of the header. When the header is a multi-line header, this field contains carriage return characters.

public int `order`
- This field is used to reassemble headers in the order they appeared in the message when it arrived.

public boolean `synthetic`
- indicates a synthetic header that didn't really appear in the message, but we're encoding some private (and immutable) data in it.

public Object `msg_fk`
- Reference to the `Message` of which this header is part.

FIELDS

public static FieldDescriptor `f_reference`
public static FieldDescriptor `f_primaryKey`
public static FieldDescriptor `f_name`
public static FieldDescriptor `f_whitespace`
public static FieldDescriptor `f_value`
public static FieldDescriptor `f_order`
public static FieldDescriptor `f_msg`
public static FieldDescriptor `f_synthetic`

CONSTRUCTORS

public `Header`(relational.Database db)

Usage Create a header in a given Database.

METHODS

public Message `getMsg`( )

Usage Retrieve the Message to which this header belongs.

public void `setMsg`(relational.email.Message m)

Usage Attach this header to a given message.

Class **HeaderPriorityComparator**

public class HeaderPriorityComparator
extends java.lang.Object
implements java.util.Comparator

A Comparator object used to sort headers to put a set of the most important headers at the top, followed by the remaining headers in their internally-specified order. This sort provides a nice at-a-glance view of a message's headers.

Methods
public int `compare`(java.lang.Object o1, java.lang.Object o2)
public static HeaderPriorityComparator `getComp`( )

Class **Mailbox**

public class Mailbox
extends java.lang.Object

A tool for importing mail from a Berkeley-style mail folder into a relational email database.

Constructors
public `Mailbox`( )

Methods
public static void `comment`(int i, java.lang.String s)

Usage Debug tool.

public static Vector `importMail`(relational.Database db, java.io.InputStream is, java.lang.String folderName)

Usage Parse mail from InputStream `is` into database `db`. The original `folderName` is attached to each message as a synthetic header to preserve the user's categorization.

public static Vector `importMail`(relational.Database db, java.lang.String filename)

Usage Import mail given a Unix filename.

public static void `main`(java.lang.String []argv)

Usage Unix command-line interface to the mail import tool.

Class **MenuPanel**

public class MenuPanel
extends javax.swing.JPanel
implements java.awt.event.ActionListener, javax.swing.event.ChangeListener

A graphical menu that can be quickly navigated with keystrokes.

CONSTRUCTORS
public `MenuPanel`(relational.email.WherePanel wp)

METHODS
public void `actionPerformed`(java.awt.event.ActionEvent e)
protected void `addSubmenus`(javax.swing.JMenu parent, relational.Where w)
public void `doPendingCommand`( )
protected void `error`(java.lang.String s)
protected void `errorNotDefined`(java.awt.event.KeyEvent e)
protected void `escapeKey`( )
public void `requestFreeText`(java.lang.String label, java.lang.Runnable runnable)
protected void `setQuery`(relational.Where query)
public void `stateChanged`(javax.swing.event.ChangeEvent e)

Class **Message**

public class Message **extends** relational.ClumpRelational

The focal point of the email schema. A message is an empty object (just a primary key). The `Body`s and `Header`s refer to a Message object together form the complete message.

FIELDS
public static FieldDescriptor `f_primaryKey`
public static FieldDescriptor `f_reference`

CONSTRUCTORS
public `Message`(relational.Database db)

Class **MessageView**

public class MessageView **extends** javax.swing.JPanel

The panel that displays a single message. Can also be configured to allow an outgoing message to be composed (edited) in the same view object.

CONSTRUCTORS
public `MessageView`( )

METHODS
public void `loadMessage`(relational.Database db, relational.email.Message m)
public void `loadMessage`(java.lang.String headers, java.lang.String body)
public void `setComposing`(boolean state)

Class **State**

public abstract class State **extends** java.lang.Object

A state of the CommandPanel state machine. Concrete subclasses are defined inside `CommandPanel`.

CONSTRUCTORS

public `State`( )

METHODS
public void `enterState`( )
public abstract void `keyTyped`(java.awt.event.KeyEvent e)

Class **SubjectComparator**

public class SubjectComparator
extends java.lang.Object
implements java.util.Comparator, java.io.Serializable

Sort headers by subject. This sort honors email conventions such as Re:, so that threads sort together with the initial message that has no Re: prefix.

CONSTRUCTORS
public `SubjectComparator`( )

METHODS
public int `compare`(java.lang.Object o1, java.lang.Object o2)
protected String `trimPunct`(java.lang.String s)
protected String `trimRe`(java.lang.String s)

Class **Test**

public class Test
extends java.lang.Object

A class for debugging the Where clauses. The sophistication of the relational database package grew as the email package asked more and more of it. Inverting queries was surprisingly tricky.

CONSTRUCTORS
public `Test`( )

METHODS
public static void `main`(java.lang.String []args)

Class **TileDisplayManager**

public class TileDisplayManager
extends java.lang.Object
implements DisplayManager

A display manager that tiles the various GUI panels together in a single window.

METHODS
public void `add`(javax.swing.JComponent c, java.lang.String title)
public FilterView `getFilterView`( )
public MessageView `getMessageView`( )
public void `registerKeyboardAction`(java.awt.event.ActionListener a, java.lang.String s, javax.swing.KeyStroke k, int c)
public void `setVisible`(boolean state)

Class **TransferableFilter**

public class TransferableFilter
extends java.lang.Object
implements java.awt.datatransfer.Transferable

A draggable entity (Transferable) that lets the user drag Filters to and from the tree view of the current query.

CONSTRUCTORS
public `TransferableFilter`(relational.Where where)

METHODS
public Object `getTransferData`(java.awt.datatransfer.DataFlavor flavor)
public DataFlavor `getTransferDataFlavors`( )
public Where `getWhere`( )
public boolean `isDataFlavorSupported`(java.awt.datatransfer.DataFlavor flavor)

Class **WherePanel**

public class WherePanel
extends javax.swing.JPanel
implements java.awt.dnd.DropTargetListener

A panel that displays a graphical (tree) representation of the query that currently defines the index view.

CONSTRUCTORS
public `WherePanel`( )

METHODS
public void `addChangeListener`(javax.swing.event.ChangeListener cl)
public void `dragEnter`(java.awt.dnd.DropTargetDragEvent e)
public void `dragExit`(java.awt.dnd.DropTargetEvent e)
public void `dragOver`(java.awt.dnd.DropTargetDragEvent e)
public void `drop`(java.awt.dnd.DropTargetDropEvent e)
public void `dropActionChanged`(java.awt.dnd.DropTargetDragEvent e)
protected void `fireChangeEvent`( )
public Where `getQuery`( )
public void `removeChangeListener`(javax.swing.event.ChangeListener cl)
public void `setQuery`(relational.Where query)

Class **WhereTreeCellEditor**

public class WhereTreeCellEditor
extends java.lang.Object
implements javax.swing.tree.TreeCellEditor

WhereTreeCellEditor.java A customized editor for my whereClause tree. swiped from O'Reilly Java Swing ch 17.

CONSTRUCTORS
public `WhereTreeCellEditor`( )

METHODS

public void `addCellEditorListener`(javax.swing.event.CellEditorListener l)
public void `cancelCellEditing`( )
public Object `getCellEditorValue`( )
public Component `getTreeCellEditorComponent`(javax.swing.JTree tree, java.lang.Object value, boolean isSelected, boolean expanded, boolean leaf, int row)
public boolean `isCellEditable`(java.util.EventObject event)
public void `removeCellEditorListener`(javax.swing.event.CellEditorListener l)
public boolean `shouldSelectCell`(java.util.EventObject event)
public boolean `stopCellEditing`( )

Class **WhereTreeModel**

public class WhereTreeModel
extends java.lang.Object
implements javax.swing.tree.TreeModel

A model that manages the WhereClause displayed in a WherePanel. Based on ExpressionTreeModel.java from Java Swing ch 17 pg 574-5 (O'Reilly)

CONSTRUCTORS
public `WhereTreeModel`(relational.Where root)

METHODS
public void `addTreeModelListener`(javax.swing.event.TreeModelListener tml)
protected void `fireTreeNodesChanged`(java.lang.Object source, java.lang.Object []path, int []ci, java.lang.Object []cc)
protected void `fireTreeStructureChanged`(java.lang.Object source, java.lang.Object []path, int []ci, java.lang.Object []cc)
public Object `getChild`(java.lang.Object node, int index)
public int `getChildCount`(java.lang.Object parent)
public int `getIndexOfChild`(java.lang.Object parent, java.lang.Object child)
public Object `getRoot`( )
public void `gratuitousRootEvent`( )
public void `insertNode`(relational.Where parent, java.lang.Object node, int index)
public boolean `isLeaf`(java.lang.Object node)
public void `refresh`(javax.swing.event.TreeExpansionEvent tee)
public void `removeTreeModelListener`(javax.swing.event.TreeModelListener tml)
public void `valueForPathChanged`(javax.swing.tree.TreePath path, java.lang.Object newValue)
protected Where `whereFor`(java.lang.Object newValue)

Class **WindowDisplayManager**

public class WindowDisplayManager
extends java.lang.Object
implements DisplayManager

A DisplayManager that organizes the GUI in separate windows. There is exactly one command window; each Message gets a new window. I think I had in mind that one could preserve existing queries by creating new queries in new windows (that display the corresponding index view of the results of the query).

METHODS
public void `add`(javax.swing.JComponent c, java.lang.String title)
public FilterView `getFilterView`( )
public MessageView `getMessageView`( )
public void `registerKeyboardAction`(java.awt.event.ActionListener a, java.lang.String s, javax.swing.KeyStroke k, int c)
public void `setVisible`(boolean state)

Package rmi

This package implements the Snowflake-over-RMI authorization protocol. It is a very simple protocol: The server rejects any request for which it cannot verify the client's authority using a `SfNeedAuthorizationException`. The exception carries information to the client about the proof it needs. The client's stubs use the `InvokeHack.invoke` method to invoke services; that method catches the SfNeedAuthorizationException, uses a Prover to prove the client's authority, and sends the proof back to the server by calling a method on the exception object itself. Then the invoke method retries the remote call; if it fails again, the exception is passed back into the client code for the application programmer to handle.

Interfaces

Interface **ProofRecipient**

public interface ProofRecipient **extends** java.rmi.Remote

The Remote interface that defines how a client communicates a proof of authority to a server. It is used by `SfNeedAuthorizationException.sendProof`.

METHODS

public void `hereIsProof`(proof.Proof proof)

Usage Give the recipient (server) a required proof of authority.

Classes

Class **InvokeHack**

public class InvokeHack **extends** java.lang.Object

This class holds a static worker method that stubs call to automatically handle Snowflake authority challenges from servers.

Ideally, I would replumb RMI to do this in UnicastRef.invoke(), but the current version of RMI makes such plumbing very difficult. I chose this approach for expediency. The necessary changes to the stub objects to use this helper method are trivial and mechanical, so it's not an unreasonable shortcut.

CONSTRUCTORS

public `InvokeHack`( )

METHODS

public static Prover2 `getCurrentProver`( )

Usage Retrieve the Prover bound to this thread that the `invoke` method would use to authorize requests.

public static Object `invoke`(java.rmi.server.RemoteRef lr, java.rmi.Remote obj, java.lang.reflect.Method method, java.lang.Object []params, long opnum)

Usage This is the static worker method. It is designed to interpose on the lr.invoke() call made by RMI stub objects. It handles authority requests (`SfNeedAuthorizationExceptions`) by consulting a Prover object bound to the current thread.

public static void `setCurrentProver`(proof.Prover2 prover)

Usage Bind a Prover to the current thread.

Class **OneLineCacheRecipient**

public class OneLineCacheRecipient
extends java.rmi.server.UnicastRemoteObject
implements ProofRecipient

A ProofRecipient that accepts a single proof per subject (client) and caches one per (server) thread, so that the next attempt at the authorized action will find the proof and succeed. It is used by a server to cache the current authority under which a client is operating.

It is not difficult to conceive of servers where a cache with more than one entry would be desirable, but the exact discipline of the cache may be application-specific. I envision a collection of cache objects that can be parameterized according to Snowflake restriction tags to define their eviction policy.

This class is notably *not* of the sort that requires an SfProof to proceed. :v)

METHODS

public static Proof `getCachedProof`(sdsi.Subject subj)

Usage The server's `checkAuth()`-style method calls this method to retrieve the cached proof for a particular subject.

public static OneLineCacheRecipient `getRecipient`( )

Usage Get the distinguished cache object. This cheesy class defines only one such object per JVM.

public void `hereIsProof`(proof.Proof proof)

Usage Give the recipient a proof of a required property. This is the Remote method called by the client (actually by the client's `InvokeHack.invoke` helper method) to transmit a proof to the server's cache (this thing).

public static void `reconfigure`(boolean state)

Usage This reconfiguration method is used by `timingexp.RMIExp` to turn proof caching on and "off," where proofs last only a single request. This mode lets me time how long it takes to transmit the actual proof.

Class **SfRemoteObject**

public class SfRemoteObject
extends java.rmi.server.UnicastRemoteObject

A remote object that knows how to use the Sf proof.Prover to automatically find and send proofs of authorization for RMI calls.

@deprecated This was an attempt to replumb RMI as I did once in the `sf.rmi` package; I eventually took the shortcut described in `InvokeHack`.

@todo NOT FOR DISTRIBUTION The code in this class is based on Sun's sun.rmi.server.UnicastRemoteObject class. I needed to tweak the functionality of a specific method that Sun declared private. To do so, I had to copy the method and tweak the code. There are ways one could imagine rewriting the class binary to accomplish the same task without distributing something very close to Sun's code.
@classSummaryOnly true

Exceptions

Interface **SfNeedAuthorizationException**

public class SfNeedAuthorizationException
extends java.lang.RuntimeException

This exception is the message sent from server to client in response to a request for which the server has no proof of the client's authority. It includes

- the identity of the required issuer principal,
- the subject that tried to make the request,
- a minimum restriction tag,
- a textual description of the error, and
- a `ProofRecipient` object where the client's proof can be sent before it retries its request. The `sendProof` method is the client's interface to this operation.

CONSTRUCTORS

public `SfNeedAuthorizationException`(sdsi.SDSIPrincipal issuer, sdsi.Subject subject, sdsi.Tag tag, rmi.ProofRecipient proofRecipient)

public `SfNeedAuthorizationException`(sdsi.SDSIPrincipal issuer, sdsi.Subject subject, sdsi.Tag tag, rmi.ProofRecipient proofRecipient, java.lang.String description)

METHODS

public SDSIPrincipal `getIssuer`( )

Usage Retrieve the identity of principal over which the proof must show authority (the *issuer*).

public Subject `getSubject`( )

Usage Retrieve the *subject* who the principal believes **says** the rejected request.

public Tag `getTag`( )

Usage Retrieve the minimum restriction set that includes the rejected request.

public void `sendProof`(proof.Proof proof)

Usage The client (`InvokeHack.invoke`) calls this method with its proof of authority to have that proof shipped to the server. The method sends a proof via RMI to a destination where it will be noticed when the call that originally caused this exception is retried.

The proof must show that $S \stackrel{T}{\Rightarrow} I$, where

S = getSubject()

I = getIssuer()

T = getTag()

public String `toString`( )

Usage Retrieve a textual description of the exception.

Package sdsi

This documentation only covers my changes to Morcos' original SPKI classes, which include the new implementations of tags and new principals that Snowflake adds to SPKI. The most significant change is my complete reimplementation of tags based on the tag semantics developed in the dissertation. It includes these classes:

- NullTagException
- RCAlpha
- RCAny
- RangeComparator
- TEByteString
- TEList
- TENull
- TEParse
- TEPrefix
- TERange
- TESDSIObject
- TESet
- TESpecial
- TEStar
- Tag
- TagExpression
- TagTest

The following classes are new "SDSIObjects" I have introduced to the package, including three new types of principal:

- PseudoPrincipal
- Quoting
- SignedCertificate
- ThresholdSubject

The following classes are documented here because they include small but semantically important changes to classes that came in Morcos' package.

- Acl
- ObjectHash
- SDSIObject
- Tuple

Classes

Class **Acl**

public class Acl
extends sdsi.SDSIObject

Changed Morcos' class to use a more meaningful intersection test than just "has a non-null intersection." See dissertation Section 6.6.

@author Alex Morcos
@author changed by jonh@cs.dartmouth.edu
@classSummaryOnly true

Class **ObjectHash**

public class ObjectHash
extends sdsi.SDSIPrincipal
implements Subject

This subject is an s-expression representing the hash of some (external?) object. Its representation is a list `(object-hash` *the-hash*`)`

which clearly indicates what the heck the hash is. [jonh]

@author jonh made this into a principal, since it can speak for others in Snowflake. Example: proof.MacProof. Also added the (Hash) constructor.
@classSummaryOnly true

Class **PseudoPrincipal**

public class PseudoPrincipal
extends sdsi.SDSIPrincipal
implements Subject

A PseudoPrincipal is not meant to be used in any real statements. Instead, it's a "space" in a prototype statement to be filled in by a recipient to make a concrete statement.

Fields

public static final String `LABEL`

Constructors

public `PseudoPrincipal`(sdsi.sexp.SexpList sexplist, sdsi.SDSIPrincipal nameContext)

Usage Parse an S-expression into a pseudo-principal.

Parameters

- `sexplist` - An SexpList to parse into a PseudoPrincipal subject.
- `nameContext` - the principal to whom any names specified in the sexplist are meaningful.

public `PseudoPrincipal`(java.lang.String description)

Usage Create a PseudoPrincipal.

Parameters

`description` - a textual description that can hint to a human what principal should take the place of this stand-in object.

METHODS

public String `getDescription`( )

Usage Return the textual description of what real principal belongs here.

public String `toShortString`( )

Class **Quoting**

public class Quoting **extends** sdsi.SDSIPrincipal

One principal quoting another. The first is actually the principal doing the speaking, the second is only named by the first; there is no intention that the second principal has any awareness or acquiescence to the statement being made.

The speaking principal also must explicitly claim to be quoting another; it is a useful mechanism for writing secure multiplexed services. The service is the quoting principal; it quotes those principals on behalf of whom it is performing its service. That way it will not accidentally perform an action for one client using its authority granted to it by another.

An extension to SPKI. Relax! Relax! Yes I know extending a security protocol is a dangerous proposition. This extension is justified by my semantic model of SPKI in my thesis. The premise of the model is to give meaning to SPKI's constructs, and this construct preserves the meaning.

This thing is a SDSIPrincipal (which really means an Issuer) because we want it to be legal in the issuer field of a certificate. That's sensical in my logic, so it's as valid an extension as any.

@todo ThresholdSubjects should really be ThresholdPrincipals for the same reason.

@author jonh@cs.dartmouth.edu

FIELDS

public static final String `LABEL`

CONSTRUCTORS

public `Quoting`(sdsi.sexp.SexpList sexplist, sdsi.SDSIPrincipal nameContext)

Parameters

`sexplist` - An SexpList to parse into a quoting subject.

`nameContext` - the principal to whom any names specified in the sexplist are meaningful.

public `Quoting`(sdsi.Subject quoter, sdsi.Subject quotee)

METHODS

public Subject `getQuotee`( )

public Subject `getQuoter`( )

public String `toShortString`( )

Class **SDSIObject**

public class SDSIObject
extends java.lang.Object
implements java.io.Serializable

Added support for parsing Quoting, PseudoPrincipal, and ThresholdSubject principals. Added performance optimizations for `equals` and `hashCode`, which were previously pretty appalling. Now they're only fairly appalling.

@todo Made `srep` and the constructors public so that I could extend it outside of this package (so my code was a *little* factored). A production implementation should clean up the package hierarchy and decide whether this class should really be public anymore.
@author Alex Morcos
@author changed by jonh@cs.dartmouth.edu
@classSummaryOnly true

Class **SignedCertificate**

public class SignedCertificate
extends sdsi.SDSIObject

This object represents the Snowflake statement:

$$A|Q_1|Q_2 \ldots \textbf{ says } (B \Rightarrow A|Q_1|Q_2| \ldots)$$

for a public-key principal A. The actual statement representation is a certificate (Cert) representing the

$$(B \Rightarrow A|Q_1|Q_2|...)$$

statement (including the auth tag I have omitted from the formula), plus a signature by A on the whole statement. Since A is free to quote anyone she wants, her "saying" the right-hand statement (by calling this ctor) is taken to mean she is quoting $Q_1 \ldots Q_n$, hence the $A|Q_1| \ldots |Q_n$ in the left argument of the 'says' operator.

A SignedCertificate is not really a SDSI object, it's more of a Snowflake object. It's a member of a superset of SDSIObject, conceptually (although not according to the way I'm abusing the class hierarchy).

Currently it lives here in sdsi.*, because SDSIObject isn't designed to be extended outside its own package.

This class may want to be extended to allow the speaker to be different than the cert issuer.

This instantiation of SignedCertificate embodies a form of the handoff rule (if you unequivocally believe verify()):

$$A \textbf{ says } C \Rightarrow A|B \supset C \Rightarrow A|B$$

The handoff rule and an implicit assumption that A's signature on the statement actually means $A|B$ **says** ... makes that so. This assumption doesn't allow the "undesirable form" of the handoff rule mentioned in Lampson, though:

$$A \Rightarrow G \wedge A \text{ **says** } B \Rightarrow G \supset B \Rightarrow G$$

So it's a pretty restricted, sensible part of the rule.

@jonh@cs.dartmouth.edu

Fields

public static final String `LABEL`

Constructors

public `SignedCertificate`(sdsi.Cert certificate, sdsi.SDSISignature signature)

Usage Construct a signed certificate from a certificate and a signature for the certificate.

public `SignedCertificate`(sdsi.sexp.SexpList l)

Usage Parse a SignedCertificate from an S-expression.

Methods

public SDSIPublicKey `getBaseSpeaker`( )

Usage Return the public key actually speaking (signing the cert); less the list of principals he's quoting.

public Cert `getCertificate`( )
public SDSISignature `getSignature`( )
public String `getType`( )
public SDSIPublicKey `unwindQuoting`(sdsi.SDSIPrincipal principal)

Usage Unwind the quoted principals from a principal quoting a chain of other principals.

public void `verify`( )

Exceptions

`proof.InvalidProofException` - if the signature is invalid

Class **Tag**

public class Tag
extends sdsi.SDSIObject

A Tag object represents an s-expression that begins (tag ...). TagExpressions are the things inside that Tag object. So a Tag object is a wrapper for the whole expression.

This class and its companion classes TagExpression, TagTest, TE*, RangeComparator, and RC* are all rewritten by jonh@cs.dartmouth.edu based on my formal semantics for tags and tag intersection in Chapter 6 of my thesis.

Fields

public static final String `LABEL`

- S-expression label for this data structure.

CONSTRUCTORS

public `Tag`(sdsi.sexp.SexpList l)

Usage Parse a S-expression into a Tag object.

METHODS

public static Tag `getNullTag`( )

Usage Get the special tag representing no authorization (an empty set, or A_{null})

public static Tag `getTagStar`( )

Usage Returns (tag *), the tag representing the set of all auths A.

public boolean `hasSubset`(sdsi.Tag otherTag)

Usage Returns true if otherTag represents a subset of the set represented by this tag. The test is performed by intersecting the two and seeing if you get back otherTag.

Comments:

1. The test is sound, but I'm not sure that it is complete. Might there be times when the thing you get back is logically equivalent to this tag, but not syntactically equivalent?
2. The original SDSI code had a `boolean intersects()` method that returned true if this Tag and `otherTag` intersect to something other than the empty set (TENull). You'd think this would be a good way to test membership of a request in a restriction set, but what if the request is a bigger set than the restriction set? That's a weird way to structure requests, but in some circumstances it may be meaningful. So we really want to test whether otherTag is a subset of this tag, hence this method.
3. if otherTag.isNull(), this test will always succeed. (And rightfully so.)

public Tag `intersect`(sdsi.Tag otherTag)

Usage Returns a tag that represents the set formed from the intersection of the sets represented by this tag and the otherTag. (Got that?) If there is "no intersection" (as the SPKI document calls it), the resulting tag will return true when asked isNull().

public boolean `isNull`( )

Usage Return true if this tag represents the null set of auths A_{null}

public String `toShortString`( )

Usage Returns the first 15ish characters of the tag representation as a String.

public Tag `union`(sdsi.Tag otherTag)

Usage Return a tag that represents the union of the auths represented by `this` tag and `otherTag`. The new tag is simply the A_{set} of the original two tags.

Class **TagTest**

public class TagTest
extends java.lang.Object

This class has a bunch of test cases to verify my implementation of Tags.

CONSTRUCTORS

public `TagTest`( )

METHODS

public static void `main`(java.lang.String []args)

Class **ThresholdSubject**

public class ThresholdSubject
extends sdsi.SDSIObject
implements Subject

The SPKI Threshold Subject – it speaks for the issuer when k of the n listed principals agree on the statement; that is, when one can prove that one speaks for k different principals of the n listed.

@author jonh@cs.dartmouth.edu

FIELDS

public static final String `LABEL`

CONSTRUCTORS

public `ThresholdSubject`(sdsi.sexp.SexpList sexplist, sdsi.SDSIPrincipal nameContext)

Usage Parse a ThresholdSubject from an S-expression.

Parameters

- `sexplist` - An SexpList to parse into a threshold subject.
- `nameContext` - the principal to whom any names specified in the sexplist are meaningful.

public `ThresholdSubject`(sdsi.Subject []subjects, int k)

Usage Build a threshold principal given existing Subjects.

Parameters

- `subjects` - the list of n Subjects. (n=subjects.length)
- `k` - the number of subjects that must agree to represent this principal

METHODS

public int `getK`( )

Usage Return the number k of principals that must agree in order that together they speak for the issuer.

public int `getN`( )

Usage Return the total number of principals named in the ThresholdSubject.

public Subject `getSubject`(int i)

public String `toShortString`( )

Class **Tuple**

public class Tuple
extends java.lang.Object

Changed Morcos' class to use a more meaningful intersection test than just "has a non-null intersection." See dissertation Section 6.6.

@author changed by jonh@cs.dartmouth.edu
@classSummaryOnly true

Exceptions

Interface **NullTagException**

public class NullTagException
extends java.lang.RuntimeException

Thrown if an operation cannot complete because its argument contains a TENull tag.

@deprecated since we now use a concrete sexp representation for this tag.

Package sdsi.sexp

Enhanced versions of Morcos' implementation of Rivest's S-expressions, an unambiguous data structure representation. I modified the packages in this class to support a much more efficient dynamic reconstruction of the default S-expression representations, which is helpful not only when you think you're outputting a S-expression, but they're used heavily in the `sdsi.SDSIObject`'s equals() and hashCode() methods. So these changes actually clean up a big chunk of inefficiency in the original code. But there's a lot more there to fix.

I also extended the SexpList interface to make it much easier for my own SDSIObject subclasses to conveniently construct SexpList objects.

Otherwise, the interface to this package is not especially fascinating; it is a worker class heavily used by the `sdsi` package, and has few interesting publicly useful methods. So I will suppress the details here.

Package servlet

This package includes servlets that implement the server-side of Snowflake HTTP authorization, including a file server and an email gateway.

Interfaces

Interface **SSLConfiguration**

public interface SSLConfiguration

An interface to allow SSLListener to extract the SSLContext from a configuration. This approach only supports one SSLContext per server. It might be better to have an array of them, as is done with the listenerClasses[] and such configs in HttpConfiguration.

METHODS
public SSLContext `getSSLContext`( )

Classes

Class **FileServlet**

public class FileServlet **extends** servlet.ProtectedServlet

FileServlet serves up a tree of files using Snowflake security implemented by ProtectedServlet. Individual requests are handled by instantiating an inner class, FileServlet.PMHandler, that holds the state of the request while its methods chew away on it.

CONSTRUCTORS
public `FileServlet`( )

METHODS
public void `init`(javax.servlet.ServletConfig config)

Usage Standard servlet initialization from a configuration object. This method creates a Prover that gathers initial delegations from the hard-coded directory `certs-server`.

The config parameter `root` defines the top of the Unix file tree to serve.

Class **MailServlet**

public class MailServlet **extends** servlet.ProtectedServlet

MailServlet serves up email messages stored in a `relational.Database` according to the schema in `relational.email`. Messages and queries are mapped into Snowflake restriction sets (SPKI tags) and protected with delegated authority.

This is the Gateway example from the thesis. It quotes the client program to ensure that the server is making access-control decisions, even if the gateway holds delegated authority over databases belonging to multiple users.

@author jonh@cs.dartmouth.edu

CONSTRUCTORS

public `MailServlet`( )

METHODS

public void `doGet`(javax.servlet.http.HttpServletRequest request, javax.servlet.http.HttpServletResponse response)

Usage doGet() handles a single request from a client. It binds the client's thread to an SSHContext associated with the gateway's identity, and instantiates an inner class to handle the individual request.

public void `init`(javax.servlet.ServletConfig config)

Usage Standard servlet initialization from parameters in a configuration object. This servlet grabs its identity from a pregenerated public key in `certs-sharedserver/`; a production implementation should generate (and cache) its own public keys or other form of principal identification.

Then it creates a Prover to cache delegations and handle demands of authority from the Database object.

Class **NaglessListener**

public class NaglessListener
extends com.mortbay.HTTP.HttpListener

A Nagle-less version of the Jetty HttpListener, to avoid timer delays that make it hard to measure HTTP performance characteristics.

Nagle's algorithm causes a packet send to be delayed by some time, in the hopes that another write will come along soon enough to be grouped into the same packet. Oddly, that delay appears even when using sockets connected to the localhost. In either case, it interferes with measurements where we want the bottleneck resource to be fully utilized.

In a production system, one might not want to use a NaglessListener since Nagle's algorithm can reduce wasteful network overhead. As long as the application code is properly buffering its output, however, such waste should not be a problem.

CONSTRUCTORS

public `NaglessListener`(com.mortbay.Util.InetAddrPort addrPort, com.mortbay.HTTP.HttpServer httpServer, int minThreads, int maxThreads, int maxIdleTimeMs)

Usage Create an HTTP listener.

METHODS

protected Socket `accept`(java.net.ServerSocket serverSocket)

Usage The only interesting thing this class does is in this method. When the listener accepts on a socket, this method sets the TcpNoDelay option to true (turns off Nagle's algorithm) before passing the socket on to other handlers to process the request arriving on it.

Class **ProtectedServlet**

public class ProtectedServlet **extends** javax.servlet.http.HttpServlet **implements** jp.SfHttpProtocol

ProtectedServlet is a parent class for servlets that want to check Snowflake/SPKI-style permissions on incoming requests.

CONSTRUCTORS
public `ProtectedServlet`( )

METHODS
public void `doGet`(javax.servlet.http.HttpServletRequest request, javax.servlet.http.HttpServletResponse response)

Usage Handle the GET and HEAD methods by building a simple web page. HEAD is just like GET, except that the server returns only the headers (including content length), not the body we write.

This method simply instantiates a `PSHandler` and passes the request there.

public static Proof `extractProof`(java.lang.String authHeader)

Usage Extract a proof from an "Authorization: SnowflakeProof " header. This is a common step in the Snowflake HTTP-with-signed-requests protocol.

public void `init`(javax.servlet.ServletConfig config)

Usage The standard servlet initialization method. This one stashes the configuration in an instance field `saveConfig` for subclasses to inspect.

Class **PSHandler**

public class PSHandler **extends** java.lang.Object **implements** jp.SfHttpProtocol

A ProtectedServlet instantiates a new PSHandler object (actually a subclass defined by a subclass of ProtectedServlet) to handle each individual request. Any state we store in the ProtectedServlet object itself is subject to simultaneous access by multiple threads. So for each request, we whip together a PSHandler object to gather per-request state in a convenient place.

METHODS
public void `doGet`( )

Usage Handle a single request. This method calls `requestIsAuthorized` to check the authority of the incoming request. If the request is authorized, it calls `servePage` to return the correct content. Otherwise, it calls `demandAuth` to send an appropriate Snowflake/HTTP Authorization demand to the client.

The request itself and the response object were stored in this object's state when the constructor was called.

public SDSIPrincipal `getRequiredIssuer`( )

Usage A method that determines the issuer principal; that is, the principal that controls the requested resource.

public void `servePage`( )

Usage Override this method to actually serve up your servlet-specific data.

Class **SSLListener**

public class SSLListener **extends** servlet.NaglessListener

An SSL listener for mortbay's Jetty webserver. By instantiating this instead of HttpListener, you get SSL sockets. This class uses claymoresystems' SSL implementation.

Originally written to provide a Java-SSL-client to Java-SSL-server performance baseline for comparison to Snowflake authorization.

@todo Adapt to implement Sf-over-SSL. That would make for a better performance comparison, plus it would make different and interesting security/performance tradeoffs relative to the signed-requests protocol.

Constructors

public `SSLListener`(com.mortbay.Util.InetAddrPort addrPort, com.mortbay.HTTP.HttpServer httpServer, int minThreads, int maxThreads, int maxIdleTimeMs)

Usage This is the only constructor that matters; that is, this is the one HttpServer calls when instantiating these critters. It's defined by HttpListener.ConstructArgs.

Methods

protected ServerSocket `newServerSocket`(com.mortbay.Util.InetAddrPort address, int acceptQueueSize)

Usage New server socket. Creates a new servers socket. May be overridden by derived class to create specialist serversockets (eg SSL).

Parameters

`address` - Address and port
`acceptQueueSize` - Accept queue size

Returns The new ServerSocket
Exceptions

`java.io.IOException` -

Class **SSLServerConfig**

public class SSLServerConfig **extends** com.mortbay.HTTP.Configure.BaseConfiguration **implements** SSLConfiguration

A Jetty Configuration class for an SSL HTTP server. It is not hard to combine this server with a regular HTTP server in a single process; see `fourServers`.

CONSTRUCTORS

public `SSLServerConfig`(Tools.Options opts)

Usage Using the command line options in `opts`, configure the SSL server handler.

METHODS

public static void `fourServers`( )

Usage Configure a set of four servers for use with `timingexp.HttpExp`. Two are SSL, two are plain; two use Jetty, two provide simple HTTP service using the inner class `SSLServerConfig.SimpleServer`.

public SSLContext `getSSLContext`( )

Usage Implements SSLConfiguration by supplying the requested SSLContext object.

I'd use the getAttributes() mechanism in HttpServer/HttpConfiguration, but it is deprecated for getProperties. I'd use the getProperties() mechanism, but it only returns strings.

public Class `listenerClasses`( )

Usage Listen with an SSLListener so the sockets are SSL sockets (To extend this config to support both Http and SSL, listen on two ports with different handlers.)

public static void `main`(java.lang.String []args)

Usage Configure an SSL server from the command line.

public void `runSimpleServer`(Tools.Options opts)

Usage Instantiate and run a simple server using the specified option array.

public void `setSSLContext`(Tools.Options opts)

Usage Establish an SSLContext according to the options.

Class **SSLServerConfig.SimpleServer**

public class SSLServerConfig.SimpleServer **extends** java.lang.Object

A very simple Java HTTP server. This gives us a baseline for the overhead associated with Jetty's sophisticated request and stream handling.

CONSTRUCTORS

public `SSLServerConfig.SimpleServer`(servlet.SSLServerConfig this$0)

METHODS

public void `run`(Tools.Options opts)

Usage Configure the server using the options, and loop, handling one request at a time.

Exceptions

Interface **StatusCodeException**

public class StatusCodeException
extends java.lang.RuntimeException

This exception is related to `jp.PageException` in that it encapsulates an exception and knows how to send that exception to the client via a response object. @todo In fact, this exception class should probably be merged with PageException.

CONSTRUCTORS

public `StatusCodeException`(int code, java.lang.String msg)

Usage Create an exception with the given code.

Parameters

- `code` - One of the SC_ constants defined in `java.servlet.HttpServletResponse`.
- `msg` - a textual error message

METHODS

public void `sendError`(javax.servlet.http.HttpServletResponse response)

Usage Send the exception as an HTTP error to the client.

Package sexp

This is jonh's manual C-to-Java translation of the C sexp code on Rivest's web site. Then I found Morcos' SDSI package which included a similar translation, and bailed out on this translation.

Perhaps this translation is worth working with later, because Morcos' code is pretty clunky. This code is no more optimized, though, and is missing many of the conversion functions already present in Morcos' code.

@author jonh@cs.dartmouth.edu

Package sf

The `sf` package includes the naming-related components of the Snowflake prototype.

@todo The choice of classname does not follow the Java standard. It should be changed to belong to a parent package such as `edu.dartmouth.cs.jonh`.

Interfaces

Interface **Container**

public interface Container **extends** Namespace

The Container interface defines how Snowflake manipulates an underlying data store to support stored objects. While objects bound into a Namespace may be stored anywhere, the objects bound into a Container Namespace are stored together. By binding other names to the objects in a Container, one can decouple object names from storage location; the Container simply represents the most concrete Snowflake name used for a resource.

@author Jon Howell

METHODS

public Object `allocate`(java.lang.String name, java.lang.String clazz)

Usage Create a new instance of Class c in the container and return a reference to the newly created Object. Calls c's no-argument constructor. Maybe later we'll have a way to pass a Constructor[] (See Tiger book p. 448).

We pass around the string name of the class because this interface may be used remotely, and Classes can't be passed remotely. Java's identification of classes by name is ugly – it introduces a new textual namespace that competes with Snowflake's. If clazz is an interface, the message is taken to mean that the Container should supply its own, suitable implementation of the requested interface. If no suitable implementation is available, the method returns ContainerException.

Returns a reference to the newly created Object.

public void `free`(java.lang.String r)

Usage Free the storage associated with the resource bound to name r in this Container.

public Object `store`(java.lang.String name, java.lang.Object o)

Usage copy Object o into this container.

Returns a reference to the copied Object.

Interface **Namespace**

public interface Namespace **extends** java.rmi.Remote

A Namespace is the basic Snowflake naming interface. It defines a remote object (so that all name bindings may be shared) that maps names in some context to resources. Namespaces can, of course, be recursively nested: a name may resolve to another Namespace. Hence "Directory" is a better name for this interface; for expository reasons, that is the name used in the dissertation.

METHODS

public void `bind`(java.lang.String name, java.lang.Object target)

Usage Bind an object to a name in this Namespace context. Unbind a name by calling bind with a null target object.

Parameters

`target` - should be Remote for the binding to be sharable by reference to other "processes;" or at least Serializable for the binding to be shared by value.

public boolean `completeList`( )

Usage Indicates whether `listAllNames` returns every name binding that may be resolved with `lookupName`.

It is not correct for listAllNames() to return a name that lookupName() throws a NamespaceException for, except for race conditions, when the name bindings have changed since the lookupName() was called.

Returns *true* if lookupName(name) throws a NamespaceException iff name is not in the results of listAllNames(), or *false* if lookupName() might resolve a name that listAllNames() doesn't return.

public Vector `listAllNames`( )

Usage List the names bound in this Namespace. If `completeList`() returns false, the list may be empty or incomplete, even though lookupName() returns objects for some names. This may be the case when the Namespace is an interface to an object whose list of bindings is invisible for size or privacy reasons, but on which the single-name lookup operation is allowable.

Returns a Vector of String names

public Object `lookupName`(java.lang.String name)

Usage Look up a single name in this namespace.

Returns a reference to the bound resource

Exceptions

`sf.NamespaceException` - if the name is not bound

public Object `lookupPath`(java.lang.String name)

Usage Parse a pathname into components, and recursively resolve each component name in the path. Notice that the Namespace server object performs this task on behalf of the client; it might be reasonable to refactor this operation for better network performance in different situations.

`NamespaceSupport.lookupPath` provides a reusable implementation of this method, since interfaces cannot define inheritable method bodies.

Parameters

`name` - a string with '/' as a path component delimiter

public Object `lookupPath`(java.util.Vector path, int cur)

Usage Recursively resolve each component name in a Vector of names.

`NamespaceSupport.lookupPath` provides a reusable implementation of this method, since interfaces cannot define inheritable method bodies.

Parameters

`path` - a Vector containing a list of names to resolve
`cur` - offset to the next name requiring resolution.

public int `version`( )

Usage Should return a new value whenever the namespace implemented by this interface changes. Reasonable implementations would be a checksum of the name=>object mappings, or a sequence number updated whenever a change is made. It is acceptable to change too often (such as changing even when the mapping has not changed), but may cause some clients to poll the object correspondingly often. Version numbers may be reused, but it's advisable that they not be reused for a long time (to reduce the likelihood that a client gets fooled by missing the intervening version numbers).

This method is the most crude way that a namespace can make its changes visible to interested parties. More efficient implementations are available by implementing `NamespaceUpdater`. Clients of namespaces that only support this `version` method may use a `NamespaceVersionUpdater` adapter object to provide an event-based interface to the polling-only Namespace.

If a version number is not available, throws NamespaceException. (This is the only circumstance in which it throws that exception.) Clients should interpret zero as such, and assume that no information is available as to whether the mapping has changed.

Interface **NamespaceListener**

public interface NamespaceListener **extends** java.rmi.Remote

An object that wants to be notified when a Namespace is updated should implement NamespaceListener. Then the object should register itself using the NamespaceUpdater interface of the Namespace or the adaptor watching the Namespace.

@author jonh

METHODS

public void `listenerRemoved`( )

Usage This listener has been forcibly removed and will no longer receive updates.

public void `namespaceEvent`(sf.NamespaceEvent e)

Usage Reports an event to the listener.

Interface **NamespaceUpdater**

public interface NamespaceUpdater
extends java.rmi.Remote

A NamespaceUpdater is an event generator that notifies listeners when the Namespace it monitors has changed. A typical usage is for a `Namespace` to implement NamespaceUpdater directly. This interface is useful for clients that display a dynamic view of a Namespace, such as a graphical window.

A `NamespaceVersionUpdater` can be used to provide event generation services for simple Namespaces that only support version-number polling.

@author jonh@cs.dartmouth.edu

Methods

public void `addNamespaceListener`(sf.NamespaceListener l)

Usage Register a listener that wishes to be notified when the Namespace changes.

public void `removeNamespaceListener`(sf.NamespaceListener l)

Usage Deregister a listener that no longer wishes notification of Namespace changes.

Interface **NSFileIfc**

public interface NSFileIfc
extends java.rmi.Remote

This class is meant to do roughly what java.io.File does, giving java code an interface to the metainformation about "files" in the Snowflake namespace. The notable difference is that it is a Remote interface, so that it is distributable.

From here, you can get an NSFileInputStream or NSFileOutputStream with which to read or write "files."

@author jonh

Methods

public boolean `canRead`( )

Usage Indicates whether this File object is readable.

public boolean `canWrite`( )

Usage Indicates whether this File object is writable.

public RemoteInputStream `getInputStream`( )

Usage Get a (remote) stream object over which bytes from this file may be read.

public RemoteOutputStream `getOutputStream`( )

Usage Get a (remote) stream object over which bytes for this file may be written.

Interface **NSInputStreamIfc**

public interface NSInputStreamIfc
extends java.rmi.Remote

A NSInputStream lets you read bytes from a remote object. You need to pass it to a RemoteInputStreamGlue to have a real java.io.InputStream, which you can then pass to something like a DataInputStream (for reading binary data) or an InputStreamReader (for text).

@deprecated This class was replaced by `ide.RemoteInputStream`, which provides the same functionality as this interface. This class was originally designed to be used by the Unix compatibility environment. Unix system calls map better to the random-access interface than to separate, stateful input and streams.
@author jonh

Methods

public int `available`( )
public void `close`( )
public int `read`( )
public NSInputStreamPacket `read`(int max)

Interface **NSRandomAccessFileIfc**

public interface NSRandomAccessFileIfc
extends java.rmi.Remote

NSRandomAccessFileIfc is a wire (Remote) version of the java.io.RandomAccessFile interface. Many methods correspond, but also throw RemoteException.

This interface is used in the UFO-based Unix-compatibility environment layer.

@author jonh@cs.dartmouth.edu

Methods

public void `close`( )
public long `getFilePointer`( )
public long `length`( )
public int `read`( )
public NSInputStreamPacket `read`(int max)

Usage The read() interface for multiple bytes changes slightly from java.io.RandomAccessFile, because we need to return an array by value, not pass an empty one in by reference.

public void `seek`(long pos)
public void `write`(byte []b, int off, int len)
public void `write`(int b)

Interface **Program**

public interface Program
extends java.rmi.Remote

Program is a simple interface that explicitly declares that a given resource is a "shell-runnable" resource. A Program takes as an argument a root Namespace, which may by convention include an `argv` directory (another Namespace bound to "argv" in the root Namespace). Programs typically also expect to find streams bound to "/stdin" and "/stdout."

This interface specifies an extant program object, with its own process. We should define another interface for a program image, a class file that gets loaded and instantiated locally, revealing a Program interface for invocation.

@author jonh

METHODS

public Object `run`(sf.Namespace root)

Usage Invoke the Program object.

Returns an arbitrary Object as a result.

Classes

Class **cat**

public class cat **extends** java.rmi.server.UnicastRemoteObject **implements** Program, java.io.Serializable

A `Program` used in the `ide.Shell` to print the textual contents of a `NSFileIfc` object.

CONSTRUCTORS

public `cat`( )

METHODS

public Object `run`(sf.Namespace root)

Class **ClassReloader**

public class ClassReloader **extends** java.lang.ClassLoader

An attack at the Class Evolution problem. (For more information, look at the proceedings of the Persistent Java Workshops.) The intention was that a HashContainer would use a ClassReloader to acquire a new class definition for a newly-allocated object when the class had changed.

@deprecated At the time, however, my interfaces were changing rapidly enough that ClassReloaded objects could not communicate with one another because they had no common interface loaded by the system classloader.

CONSTRUCTORS

public `ClassReloader`(java.lang.String myclass)

METHODS
public synchronized Class `loadClass`(java.lang.String name, boolean resolve)
public byte `loadClassData`(java.lang.String name)

Class **ContainerServer**

public class ContainerServer **extends** java.lang.Object

Build a low-level resource that implements a Container object from a Unix JVM process. From the Snowflake point of view, a user operating as a system administrator invokes this class to "create" a resource from lower-level raw resources (a "disk" that speaks POSIX :v). The Container server binds this resource into Java RMI's flat name registry. Then the administrator binds this resource (using the "low-level" RMIRegistry name for the resource) into his own Snowflake namespace. From that point on, the resource is only manipulated using Snowflake names, as the administrator uses it and shares it with others.

CONSTRUCTORS
public `ContainerServer`( )

METHODS
public static void `main`(java.lang.String []args)

Usage The Unix (java interpreter) command-line interface to create this object and bind it in the RMIRegistry. An optional parameter determines the RMIRegistry low-level name at which the resource is bound. That name is used by the administrator to "find" the resource when importing it into Snowflake with `mkrem`.

public void `startup`( )

Usage A worker method that creates the container, registers it in the host's RMIRegistry, and registers a callback with Icee, if available, to ensure that the container object re-exports itself after a failure recovery.

Class **ContainerServer.Reregister**

public class ContainerServer.Reregister **extends** java.lang.Object **implements** Icee.Auto.Callback

Reregister is a worker class that catches callbacks from Icee and ensures that the container's low-level RMIRegistry name is always available, even after an Icee recovery. This method re-exports the object and rebinds it to its RMIRegistry name, ensuring that any RemoteStubHacks will be able to successfully re-discover the restored resource transparently.

CONSTRUCTORS
public `ContainerServer.Reregister`(sf.ContainerServer this$0)

METHODS
public void `recovered`( )

Class **cp**

public class cp
extends java.rmi.server.UnicastRemoteObject
implements Program, java.io.Serializable

A `Program` used in the `ide.Shell` to copy an `NsFileIfc` object from one Container to another. It works the obvious way, by reading the stream from one object and writing it to the other. Clearly there is room for a more sensible implementation.

CONSTRUCTORS

public `cp`( )

METHODS

public Object `run`(sf.Namespace root)

Usage Usage is printed when run with no arguments.

Class **HashContainer**

public class HashContainer
extends sf.HashNS
implements Program, Container

A basic Container implementation. Stored objects merely live in the current virtual memory; if they are persistent, it is because Icee makes the entire JVM persistent.

CONSTRUCTORS

public `HashContainer`( )

METHODS

public Object `allocate`(java.lang.String name, java.lang.String clazz)

Usage Allocate a new instance of a given class in this Container. The class' no-argument constructor is called to instantiate it. If clazz is an interface, the message is taken to mean that the Container should supply its own, suitable implementation of the requested interface. If no suitable implementation is available, the method returns ContainerException. This class only knows about the interface `sf.NSFileIfc`.

public void `bind`(java.lang.String name, java.lang.Object o)

Usage Disallow explicit binds to the Container. That is, conventionally, the only names bound into a container object's Namespace are those objects actually stored in the container.

public void `free`(java.lang.String r)

Usage Discard the reference to the object bound to name `r`, and free the associated space. This Container implementation defers deallocation to the Java garbage collector.

public Object `lookupName`(java.lang.String name)

public Object `run`(sf.Namespace root)

Usage The shell "command-line" interface for configuring a HashContainer object.

public Object `store`(java.lang.String name, java.lang.Object o)

Usage Store an existing object in this Container. By convention, the stored object `o` should be a Serializable (not Remote) object, so that it is copied by value into the backing store of this Container. This implementation, however, makes no effort to enforce those semantics.

Class **HashNS**

public class HashNS **extends** sf.rmi.UnicastRemoteObject **implements** Namespace, NamespaceUpdater

The most-basic implementation of Namespace, this object binds names to arbitrary Remote objects. Since the objects are Remote, the bindings are always by reference. The bindings are stored in a hash table for scalability.

@author jonh

Constructors

public `HashNS`( )

Usage Create a new Hashtable-based Namespace.

Methods

public void `addNamespaceListener`(sf.NamespaceListener l)

Usage This basic implementation includes support for namespace event listeners. (Perhaps I should factor this support out into a subclass, so that HashNS is more basic?)

public void `bind`(java.lang.String name, java.lang.Object o)
public boolean `completeList`( )

Usage Since this Namespace returns only bindings stored in its hashtable, its listAllNames() method always returns a complete list.

public Vector `listAllNames`( )
public Object `lookupName`(java.lang.String name)
public Object `lookupPath`(java.lang.String name)

Usage Defers implementation to helper class NamespaceSupport.

public Object `lookupPath`(java.util.Vector path, int cur)

Usage Defers implementation to helper class NamespaceSupport.

public void `removeNamespaceListener`(sf.NamespaceListener l)
public int `version`( )

Usage Versioning is implemented by incrementing a sequence number on each bind.

Class **ln**

public class ln
extends java.rmi.server.UnicastRemoteObject
implements Program, java.io.Serializable

A `Program` used in the `ide.Shell` to establish `Symlinks`.

CONSTRUCTORS
public `ln`( )

METHODS
public Object `run`(sf.Namespace root)

Usage Usage is specified when called with no arguments.

Class **ls**

public class ls
extends java.rmi.server.UnicastRemoteObject
implements Program, java.io.Serializable

A `Program` used in the `ide.Shell` to list the visible bindings in a Namespace (directory).

CONSTRUCTORS
public `ls`( )

METHODS
public Object `run`(sf.Namespace root)

Class **mkrem**

public class mkrem
extends java.rmi.server.UnicastRemoteObject
implements Program, java.io.Serializable

A `Program` used in the `ide.Shell` to import raw resources into the Snowflake naming environment. See `ContainerServer` for a more detailed description of how raw resources are imported.

CONSTRUCTORS
public `mkrem`( )

METHODS
public Object `run`(sf.Namespace root)

Usage Usage is supplied when invoked with no arguments. The `container_URL` command-line argument is the RMIRegistry name of the raw resource.

Class **NamespaceEvent**

public class NamespaceEvent
extends java.lang.Object

A NamespaceEvent object is sent to `NamespaceListeners` when a Namespace they are attending to has changed. The event encodes information about the nature of the change to the Namespace.

@author jonh

FIELDS

public static final int `VAGUE`

- Something happened, _every_ name could have changed for all you know. (Both name and mapping are useless.) NamespaceVersionUpdater sends this message, since it does not inspect the Namespace's list at each change to discover the differences.

public static final int `NAME`

- The enclosed name changed; but no mapping is supplied. Receiver must query Namespace to discover the mapping if desired.

public static final int `MAPPING`

- Both name and mapping are valid (most explicit event)

public int `type`

- One of VAGUE, NAME, or MAPPING.

public String `name`

- The name that has been changed (if type!=VAGUE)

public Object `mapping`

- The new mapping for the name (if type==MAPPING)

CONSTRUCTORS

public `NamespaceEvent`(int type)

public `NamespaceEvent`(int type, java.lang.String name)

public `NamespaceEvent`(int type, java.lang.String name, java.lang.Object mapping)

METHODS

public String `toString`( )

Class **NamespaceListenerAdapter**

public class NamespaceListenerAdapter
extends java.rmi.server.UnicastRemoteObject
implements NamespaceListener

The NamespaceListenerAdapter is a convenience superclass used by classes implementing the NamespaceListener interface. Simply override the method you are interested in receiving; invocations on the others will be ignored.

@author jonh@cs.dartmouth.edu

CONSTRUCTORS

public `NamespaceListenerAdapter`( )

METHODS

public void `listenerRemoved`( )

Usage This listener has been removed from the namespace's update list; do not expect any further events.

public void `namespaceEvent`(sf.NamespaceEvent e)

Usage A namespace of interest has changed.

Class **NamespaceSupport**

public class NamespaceSupport **extends** java.lang.Object

The NamespaceSupport class is a tool to assist Namespace implementations. It provides common parsing routines.

@todo Turn this into a convenience superclass, so that Namespace implementations inherit its functionality, rather than needing to call it.

Constructors

public `NamespaceSupport`( )

Methods

public static Vector `enumerationToVector`(java.util.Enumeration e)

Usage Extract the contents of an Enumeration as a Vector. Another one of those important little details nature, er, Sun forgot.

public static Object `lookupPath`(java.lang.String name, sf.Namespace top)

Usage Parse a path name and resolve it, beginning at the specified Namespace.

Parameters
- `top` - usually the Namespace using this class for support.

public static Object `lookupPath`(java.util.Vector path, int cur, sf.Namespace top)

Usage Look up a path beginning at the specified namespace. The path is specified as a Vector of component names plus a starting index into the vector. This method is the recursive formulation of lookupPath.

Parameters
- `path` - Vector containing a list of pathname components to resolve.
- `cur` - offset into the vector to find the component that should be resolved in `top`.
- `top` - Namespace at which to begin the resolution.

public static Vector `parsePath`(java.lang.String path)

Usage Parse a path into component pathnames. Returns a vector which is the decomposition of String path around slashes ('/'). If the first argument is an empty string, the path started with a slash, and hence was root-based (which may or may not be a meaningful distinction, depending on context). Other empty strings should be treated as NOPs.

Class **NamespaceUpdateMulticaster**

public class NamespaceUpdateMulticaster **extends** java.lang.Object

A NamespaceUpdateMulticaster is a helper object that manages a list of event subscribers and broadcasts events. It can be used to supply an implementation of the `NamespaceUpdater` interface.

@author jonh

Constructors

public `NamespaceUpdateMulticaster`( )

Methods

public void `addNamespaceListener`(sf.NamespaceListener l)

Usage Forward requests add requests on the NamespaceUpdater interface to this method to manage the subscriber list.

public void `listenerRemoved`( )

Usage Forcibly unsubscribe a listener.

public void `namespaceEvent`(sf.NamespaceEvent ev)

Usage Broadcast a NamespaceEvent to all subscribed listeners.

public void `removeNamespaceListener`(sf.NamespaceListener l)

Usage Forward requests remove requests on the NamespaceUpdater interface to this method to manage the subscriber list.

public int `size`( )

Returns a count of the current number of subscribers.

Class **NamespaceVersionUpdater**

public class NamespaceVersionUpdater
extends java.rmi.server.UnicastRemoteObject
implements NamespaceUpdater, java.lang.Runnable

This class polls a namespace's version() method, and when it changes, generates NamespaceEvents for listeners. It should be used like this:

```
NamespaceVersionUpdater nvu = new NamespaceVersionUpdater(500);
// poll twice a second
Thread nvut = new Thread(nvu);
nvut.start();
```

@author jonh

Constructors

public `NamespaceVersionUpdater`(sf.Namespace ns, long pollPeriod)

Usage Create a new NamespaceVersionUpdater to watch a Namespace.

Parameters

- `ns` - the Namespace to watch
- `pollPeriod` - how frequently to query the Namespace's version() method, in milliseconds

Methods

public synchronized void `addNamespaceListener`(sf.NamespaceListener l)

public synchronized void `removeNamespaceListener`(sf.NamespaceListener l)

public void `run`( )

Usage Runnable implementation polls Namespace and sleeps between polls.

Class **NSInputStreamImpl**

public class NSInputStreamImpl
extends java.rmi.server.UnicastRemoteObject
implements NSInputStreamIfc

An NSInputStream lets you read bytes from a remote object. You need to pass it to a RemoteInputStreamGlue to have a real java.io.InputStream, which you can then pass to something like a DataInputStream (for reading binary data) or an InputStreamReader (for text). So there could be several server-side implementations of this interface, couldn't there? Depending on whether the data source is a Unix file, or something else...

@deprecated Replaced by `ide.RemoteInputStream`.

CONSTRUCTORS

public `NSInputStreamImpl`(java.lang.String unixpath)

METHODS

public int `available`( )
public void `close`( )
public int `read`( )
public NSInputStreamPacket `read`(int max)

Class **NSInputStreamPacket**

public class NSInputStreamPacket
extends java.lang.Object
implements java.io.Serializable

See `NSRandomAccessFileIfc.read(int)` for details on how this class is used.

CONSTRUCTORS

public `NSInputStreamPacket`( )

Class **NSMemoryFileImpl**

public class NSMemoryFileImpl
extends java.rmi.server.UnicastRemoteObject
implements NSFileIfc

NSMemoryFileImpl is derived from NSUnixFileImpl. Just an object with a File-like interface, and a way to get an associated input or output stream. Not explicitly stored anywhere but memory; but perhaps made persistent with Icee. The point is to provide an alternative to files backed by the Unix filesystem. From here, you can get an `ide.RemoteInputStream` or `ide.RemoteOutputStream` with which to read or write "files," or a NSRandomAccessFileIfc object with an NFS-like interface.

@todo This is a really dumb implementation – every time the file grows, it gets copied and the old block deallocated. A simple optimization that would help most of the time would be to keep a vector of (say) 8k blocks, and tack on new ones when a write grows the file. Basically just like a filesystem would.

@classConcise true

CONSTRUCTORS

public `NSMemoryFileImpl`( )

Class **NSMemoryFileImpl.MemInputStream**

public class NSMemoryFileImpl.MemInputStream **extends** java.rmi.server.UnicastRemoteObject **implements** ide.RemoteInputStream

The implementation of `ide.RemoteInputStream` for `NSMemoryFileImpl`s.

@classConcise true

Serializable Fields

private final NSMemoryFileImpl `this$0`

Class **NSMemoryFileImpl.MemOutputStream**

public class NSMemoryFileImpl.MemOutputStream **extends** java.rmi.server.UnicastRemoteObject **implements** ide.RemoteOutputStream

The implementation of `ide.RemoteOutputStream` for `NSMemoryFileImpl`s.

@classConcise true

Serializable Fields

private final NSMemoryFileImpl `this$0`

Class **NSRandomAccessFileImpl**

public class NSRandomAccessFileImpl **extends** java.rmi.server.UnicastRemoteObject **implements** NSRandomAccessFileIfc

NSRandomAccessFileImpl is an implementation for serving up random access files to Java RMI clients. This implementation is used in the UFO-based Unix-compatibility environment layer.

@author jonh@cs.dartmouth.edu
@classConcise true

Constructors

public `NSRandomAccessFileImpl`(java.lang.String unixpath, java.lang.String mode)

Class **NSUnixFileImpl**

public class NSUnixFileImpl **extends** java.rmi.server.UnicastRemoteObject **implements** NSFileIfc

This class is meant to do roughly what java.io.File does, giving java code an interface to the metainformation about “files” in the Snowflake namespace. (Except that this class is Remote.)

From here, you can get an `ide.RemoteInputStream` or `ide.RemoteOutputStream` with which to read or write “files.”

@classConcise true

CONSTRUCTORS
public `NSUnixFileImpl`(sf.Namespace node, java.lang.String unixpath)
public `NSUnixFileImpl`(sf.Namespace root, java.lang.String path, java.lang.String unixpath)
public `NSUnixFileImpl`(sf.Namespace root, java.util.Vector path, java.lang.String unixpath)

METHODS
public NSRandomAccessFileIfc `openNSRandomAccessFileIfc`(java.lang.String mode)

Usage The Unix-emulation code uses this method to get an NFS-like interface on the file-typed object, rather than a Java-style InputStream/OutputStream interface.

Class **Proxy**

public class Proxy **extends** java.rmi.server.UnicastRemoteObject **implements** Program, Namespace

Proxy: redirects namespace requests to another namespace A proxy is like a hard link, in that its effect is invisible to the client. It is unlike a hard link in that referential integrity is not enforced.

@classConcise true

CONSTRUCTORS
public `Proxy`( )

METHODS
public Object `run`(sf.Namespace root)

Usage The shell "command-line" interface for configuring a new Proxy object.

Class **SecureContainerServer**

public class SecureContainerServer **extends** java.lang.Object

This class represents an early attempt at looking at securing resources. It portends the horrible troubles one might have when attempting to secure resources without any meaning or semantics. (*grin*)

@deprecated Since Snowflake has a legitimate security story.

CONSTRUCTORS
public `SecureContainerServer`( )

METHODS
public static void `main`(java.lang.String []args)
public void `startup`( )

Class **Sf**

public class Sf **extends** java.lang.Object

The client-side naming toolkit. Performs naming operations in the current Namespace context. A Namespace context is a "stack" of names per thread.

CONSTRUCTORS

public `Sf`( )

METHODS

public static Namespace `currentNamespace`( )

Usage Return the current Namespace – the one at the top of the current thread's namespace stack.

public static Object `lookupPath`(sf.Namespace ns, java.util.Vector nameVector, int depth)

Usage Resolve a path given the current naming context. (Recursive formulation; not typically used by clients.) This method detects Symlinks and re-resolves them. It also annotates those remote references that support name annotations (see `sf.rmi.RemoteStubHack`) for automatic name re-resolution.

public static Object `lookupPath`(java.lang.String name)

Usage Resolve a path given the current naming context.

public static Namespace `popNamespace`( )

Usage End the scope of a current Namespace declaration. Typically used in a `finally` { } block.

public static void `println`(sf.Namespace root, java.lang.String s)

Usage Print a message on the standard output stream defined by the given root namespace at `streams/output`.

public static void `pushNamespace`(sf.Namespace n)

Usage Push a new Namespace onto the thread Namespace stack. A pushNamespace operation defines the root naming context for all calls inside the scope of this declaration, until the corresponding popNamespace. A

```
try {
  ...
} finally{
  ...
}
```

block can be used to give this context the feel of a language-scoped structure.

public static void `pushNamespace`(java.lang.Thread t, sf.Namespace n)

Usage A mechanism for establishing a child thread's root Namespace. TODO: Having this being a 'public' method is a security hole if a JVM contains mutually-untrusting processes. But this mechanism is in place until Java provides a mechanism for inheritance of state from a parent thread.

public static Vector `sort`(java.util.Vector list)

Usage Still another method that is (was) mysteriously missing from the Java standard libraries.

public static BufferedReader `stdin`(sf.Namespace root)

Usage Return a reference to the standard input stream bound into the given root namespace at `streams/input`.

public static PrintWriter `stdout`(sf.Namespace root)

Usage Return a reference to the standard output stream bound into the given root namespace at `streams/stdout`.

public static String `v2s`(java.util.Vector nameVector)

Usage Turn a Vector of component names into a string pathname beginning with /.

Class **Symlink**

public class Symlink
extends java.lang.Object
implements java.io.Serializable

Symlink: A symbolic link. It is a token object that carries a new name that Sf, the client-side support library, should automatically re-resolve on behalf of the client program.

Serializable Fields
public String `target`

- The path name to be re-resolved. If it is absolute, the resolution should begin at the client's active root. If the path is relative, resolution begins at the same directory (Namespace) where this object was found.

Constructors
public `Symlink`( )

Class **Union**

public class Union
extends java.rmi.server.UnicastRemoteObject
implements Program, Namespace

A Union directory (Namespace) unions the contents of several other Namespaces. Implemented like Proxy, but with layers of "mounted" directories that operations can fall through to.

@classConcise true

CONSTRUCTORS

public `Union`( )

METHODS

public Object `run`(sf.Namespace root)

Usage The shell "command-line" interface for configuring a Union object. This is currently the only interface for adding new layers to the Union. Use the "target" command to add a layer.

Class **UnixContainer**

public class UnixContainer **extends** sf.HashNS **implements** Program, Container

A Container whose backing store is a file in a Unix filesystem. It can only store objects that implement `NsFileIfc`, for the obvious reason.

CONSTRUCTORS

public `UnixContainer`( )

Usage Instantiate a container, using the root of the Unix filesystem as the backing store.

public `UnixContainer`(java.lang.String path)

Usage Instantiate a container, using the given Unix file path as the backing store.

METHODS

public void `addNamespaceListener`(sf.NamespaceListener l)

public Object `allocate`(java.lang.String name, java.lang.String clazz)

Usage Allocate a new object stored in this Container.

Parameters

`clazz` - must be one of `UnixContainer`, indicating a new subdirectory, or `NSFileIfc`, indicating a new file.

public void `bind`(java.lang.String name, java.lang.Object o)

Usage Disallow explicit binds to the Container. That is, conventionally, the only names bound into a container object's Namespace are those objects actually stored in the container.

public void `free`(java.lang.String name)

Usage Free the resource bound to `name`. This method will attempt to delete the corresponding file or directory in the Unix filesystem.

public Vector `listAllNames`( )

public Object `lookupName`(java.lang.String name)

public Object `lookupPath`(java.util.Vector path, int cur)

public void `removeNamespaceListener`(sf.NamespaceListener l)

public Object `run`(sf.Namespace root)

Usage The shell "command-line" interface for configuring a UnixContainer object.

public Object `store`(java.lang.String name, java.lang.Object o)
public int `version`( )

Usage Change detection on a UnixContainer is implemented by inspecting the Unix modification time of the backing directory.

Exceptions

Interface **ContainerException**

public class ContainerException **extends** java.lang.Exception

A ContainerException indicates a failure occurred in handling a Container message.

Constructors
public `ContainerException`( )
public `ContainerException`(java.lang.String s)

Interface **NamespaceException**

public class NamespaceException **extends** java.lang.Exception

Something failed when performing a Namespace interface operation.

Constructors
public `NamespaceException`( )
public `NamespaceException`(java.lang.String s)

Package sf.rmi

The `sf.rmi` package includes my replumbing of RMI to support two Snowflake features: self-rebinding remote stubs that recover their bindings after losing a connection to the server, and a first hack at security based on a very early version of the speaks-for-regarding calculus. The latter feature is deprecated, since it is replaced by the newer, SPKI-based security model. Those deprecated classes are omitted.

@todo The choice of classname does not follow the Java standard. It should be changed to belong to a parent package such as `edu.dartmouth.cs.jonh`.

Classes

Class **DeputyImpl**

public class DeputyImpl
extends sf.rmi.UnicastRemoteObject
implements Deputy

This was the class that generated proofs, the analog to the current `proof.Prover`.

@deprecated Part of a prior attempt at security, before I had completely developed the logical formalism and restarted my implementation based on SPKI.

CONSTRUCTORS

public `DeputyImpl`(sf.rmi.SshEndpoint ep)

METHODS

public void `addProof`(sf.rsec.Proof proof)
public boolean `doneProof`(sf.rsec.Proof old)
public Proof `findProof`(sf.rsec.Statement s)

Class **RemoteStubHack**

public abstract class RemoteStubHack
extends java.rmi.server.RemoteStub

A variation on RemoteStub to allow external code (UnicastRef) to call getRef() so it can ask the ref to get a new channel and a new connection, when the old one is broken. Used to enable automatic resource rebinding through automatic re-resolution of names.

@todo A non-prototype implementation would need to be careful with regards to JVM security in exposing this information; perhaps “” (Java’s mysteriously unnamed default permission) is an appropriate way to control access to the reference.

CONSTRUCTORS

protected `RemoteStubHack`( )

Usage This class is accepted wherever fine RemoteStubs are also accepted.

protected `RemoteStubHack`(java.rmi.server.RemoteRef ref)

Usage This class is accepted wherever fine RemoteStubs are also accepted.

METHODS

public RemoteRef `getRef`( )

Usage Let `sf.rmi.Unicast` access this stub’s RemoteRef object.

Class **UnicastRef**

public class UnicastRef
extends java.lang.Object
implements java.rmi.server.RemoteRef, java.io.Serializable

The purpose of modifying this class is to cause remote stubs to automatically try to reconnect to their servers when the connection is lost. This step is the first in automatic rebinding. The second step is to re-resolve the name that produced the resource, and the steps beyond are to recursively re-resolve parent names in the path that arrived at this resource. A version of a Sun class modified for my nefarious purposes. I don't think I actually used this class other than temporarily while debugging the flow of RMI transactions. Plumbing.

@author modified by jonh@cs.dartmouth.edu
@todo NOT FOR DISTRIBUTION
@classSummaryOnly true

Class **UnicastRemoteObject**

public class UnicastRemoteObject **extends** sf.rmi.RemoteServer

I modified this class to see how to get it to do snowflake references that can look up stashed names in case a LiveRef fails. Basically, this class is required to instantiate `sf.rmi.UnicastServerRef`s instead of the original UnicastServerRefs. That this class is required is an artifact of RMI's current non-extensibility.

A version of a Sun class modified for my nefarious purposes. I don't think I actually used this class other than temporarily while debugging the flow of RMI transactions. Plumbing.

@author modified by jonh@cs.dartmouth.edu
@todo NOT FOR DISTRIBUTION
@classSummaryOnly true

Package sf.rsec

The `sf.rsec` package was my second hack at security, and my first implementation of an early version of the speaks-for-regarding calculus. The calculus was essentially the same as it ended up in the dissertation. This implementation is not based on SPKI, however, and has RSA keys wired in a little more tightly than my generalization of SPKI.

This implementation contains seventeen classes, but since they have been superseded by the newer implementation, this manual omits them. Their functionality is replicated in the packages listed in the *@deprecated* tag.

@todo The choice of classname does not follow the Java standard. It should be changed to belong to a parent package such as `edu.dartmouth.cs.jonh`.
@deprecated replaced by the `proof` package and new classes in the `sdsi` package.

Package sf.sec

The `sf.sec` package was my very first stab at a security model for Snowflake. It has a basic notion of restriction (`RestrictMask`), and was beginning to think about delegation (`Prinicpal`). It was replaced by the `sf.rsec` package, and then later the SPKI-based security that is documented in my dissertation. This implementation contains nine classes, but since they have been superseded by the newer implementation, this manual omits them.

@todo The choice of classname does not follow the Java standard. It should be changed to belong to a parent package such as `edu.dartmouth.cs.jonh`.

@deprecated replaced by the `proof` package and new classes in the `sdsi` package.

Package ssh

My own Java implementation of version 1 of the SSH protocol. It is "inspired" by the source code to the C code for ssh 1.x, but reorganized and rewritten and ported enough that I can claim the copyright on this version.

Some of the classes in this package add support for using SSH to protect RMI connections, as Snowflake does.

This package predates the Java Cryptography Extensions. It should be modified to merge with JCE interfaces. Specifically, my stub `SshRandom` class should be replaced with calls to a cryptographically-strong source of randomness, and the `ssh.RSA` package should become a JCE provider so that my implementation can be replaced with alternative implementations.

@author jonh@cs.dartmouth.edu

Interfaces

Interface **KeyedSocket**

public interface KeyedSocket

A Socket that implements this interface can identify the other end of the socket by its public key; it has somehow (generally by checking a signature during key exchange) shown that messages emerging from the socket on this end are spoken for by the public key returned by `getOppositeKey`.

METHODS

public RSAKey `getOppositeKey`( )

Usage Return the public key that speaks for messages read from the local end of the socket.

Interface **SRPConstants**

public interface SRPConstants

This interface puts constant definitions into the scope of any class that 'implements' it. SRP is short for Secure RMI Protocol, by which I mean RMI-over-ssh.

@author Jon Howell <jonh@cs.dartmouth.edu>

FIELDS

public static final int `SRP_CMSG_BORROW_SESSION_KEY`
public static final int `SRP_CMSG_KEY_EXCHANGE`
public static final int `SRP_SMSG_SUCCESS`
public static final int `SRP_SMSG_SERVER_KEY`
public static final int `SRP_CMSG_CLIENT_KEY`
public static final int `SRP_SMSG_SESSION_KEY`
public static final int `SRP_CMSG_SUCCESS`
public static final byte `SRP_CIPHER_IDEA`

Classes

Class **Authenticator**

public abstract class Authenticator **extends** java.lang.Object

Purpose: An instance of a subclass of this abstract class can engage the ssh server in an authentication dialog. Used to plug in different user-authentication mechanisms.

Source: for ssh protocol definition: draft-ylonen-ssh-protocol-00.txt

@author Jon Howell <jonh@cs.dartmouth.edu>

CONSTRUCTORS

public `Authenticator`( )

METHODS

public abstract void `authenticate`(ssh.BinaryPacketInputStream binaryIn, ssh.BinaryPacketOutputStream binaryOut)

Class **BinaryPacketIn**

public class BinaryPacketIn
extends java.io.DataInputStream

This class represents an incoming ssh binary packet. It's a subclass of DataInputStream, so you can pick out the packet fields using the usual methods. This class also defines some methods relevant to binary packets, such as ones that extract multiple-precision integers in ssh format.

A BinaryPacketIn object handles the type and data parts of an ssh binary packet.

Source: draft-ylonen-ssh-protocol-00.txt, page 3

@author Jon Howell <jonh@cs.dartmouth.edu>

CONSTRUCTORS

public `BinaryPacketIn`(java.io.InputStream is, int length, byte []body)

METHODS

public int `getType`( )
public BigInteger `readBigInteger`( )
public String `readString`( )
public byte `readStringAsBytes`( )

Class **BinaryPacketInputStream**

public class BinaryPacketInputStream
extends java.lang.Object

This class reads an ssh binary packet protocol stream, and produces BinaryPacketIn objects representing each packet.

Source: draft-ylonen-ssh-protocol-00.txt, pages 3-4.

@author Jon Howell <jonh@cs.dartmouth.edu>

FIELDS

public DataInputStream `dataIn`

CONSTRUCTORS

public `BinaryPacketInputStream`(java.io.InputStream i)

METHODS

public void `close`( )
public BinaryPacketIn `readPacket`( )
public void `setCipher`(ssh.Cipher cipher)

Class **BinaryPacketOut**

public class BinaryPacketOut **extends** java.io.DataOutputStream

Objects of this class are DataOutputStreams so they can be conveniently packed with data; then they are passed to a BinaryPacketOutputStream to be sent over an ssh binary packet stream.

A BinaryPacketOut object handles the type and data parts of an ssh binary packet.

Source: draft-ylonen-ssh-protocol-00.txt, pages 3-4.

@author Jon Howell <jonh@cs.dartmouth.edu>

Constructors

public `BinaryPacketOut`(java.io.ByteArrayOutputStream os)

Methods

public static BinaryPacketOut `newBinaryPacketOut`( )
public void `setType`(int type)
public byte `toByteArray`( )
public void `writeBigInteger`(java.math.BigInteger bi)
public void `writeString`(java.lang.String str)
public void `writeStringAsBytes`(byte []b, int off, int len)

Class **BinaryPacketOutputStream**

public class BinaryPacketOutputStream **extends** java.lang.Object

This class writes an ssh binary packet protocol stream, consuming BinaryPacketOut objects representing each packet.

Source: draft-ylonen-ssh-protocol-00.txt, pages 3-4.

@author Jon Howell <jonh@cs.dartmouth.edu>

Constructors

public `BinaryPacketOutputStream`(java.io.OutputStream o)

Methods

public BinaryPacketOut `newPacket`( )
public void `setCipher`(ssh.Cipher cipher)
public void `writePacket`(ssh.BinaryPacketOut op)

Class **Cipher**

public abstract class Cipher **extends** java.lang.Object

Cipher is an abstract class that defines the interface to an object that provides encipherment services.

@todo replace with JCE interfaces

Methods

public abstract void `decipher`(byte []dest, byte []src, int len)
public abstract void `encipher`(byte []dest, byte []src, int len)
public abstract void `setKey`(byte []key)

Class **CipherIdea**

public class CipherIdea **extends** ssh.Cipher

CipherIdea – implements IDEA cipher Inspired by idea.[ch] in ssh-1.2.22 (hence related variable and function names), but implemented by Jon Howell. Comments that relate to the algorithm are also verbatim from the C source.

@todo reimplement IDEA from some public document, like a paper

Constructors

public `CipherIdea`( )

Methods

public void `decipher`(byte []dest, byte []src, int len)
public void `destroyContext`( )
public void `encipher`(byte []dest, byte []src, int len)
public void `setKey`(byte []key)

Class **ClientProtocol**

public class ClientProtocol **extends** java.lang.Object

This class implements the client side of the ssh protocol. It establishes an ssh session on a channel, and then provides an InputStream/OutputStream abstraction to allow the caller to transmit data securely over the underlying channel.

Source: draft-ylonen-ssh-protocol-00.txt

@author Jon Howell <jonh@cs.dartmouth.edu>

Fields

public static String `clientVersion`

Constructors

public `ClientProtocol`( )

Methods

public void `authenticate`( )
public void `connect`(java.net.Socket socket, ssh.Authenticator []auth)
public void `connect`(java.lang.String host, ssh.Authenticator auth)

public void `connect`(java.lang.String host, int port, ssh.Authenticator auth)
public void `connect`(java.lang.String host, int port, ssh.Authenticator []auth)
public InputStream `getInputStream`( )
public OutputStream `getOutputStream`( )
public static void `main`(java.lang.String []args)
public void `preparatory`(boolean getPty)

Class **PasswordAuthenticator**

public abstract class PasswordAuthenticator
extends ssh.Authenticator

This class is an Authenticator that simply sends a password over the encrypted channel.

Source: for ssh protocol definition: draft-ylonen-ssh-protocol-00.txt especially pages 13, 21.

@author jonh@cs.dartmouth.edu

Constructors
public `PasswordAuthenticator`(java.lang.String password)

Methods
public void `authenticate`(ssh.BinaryPacketInputStream binaryIn, ssh.BinaryPacketOutputStream binaryOut)

Class **Protocol**

public class Protocol
extends java.lang.Object

This class defines the constants used in ssh version 1 protocol packets.

Source: draft-ylonen-ssh-protocol-00.txt

@author Jon Howell <jonh@cs.dartmouth.edu>
@classSummaryOnly true

Class **Protocol2**

public class Protocol2
extends java.lang.Object

This class defines the constants used in ssh protocol packets for ssh version 2. Sadly, they're totally different than ssh version 1, to the point that SSH Inc's idea of interoperability is to fire up the v.1 executable when needed to talk to a v1 remote end.

So I'm not actually implementing v2 at all. I have an implementation of v1 in here, but my main use of ssh, in Jon's "SRP" (Secure RMI Protocol) is something that looks like v1, but with some of my own messages, and v2-style channels. Source: ssh 2.0.13/lib/sshproto/sshmsgs.h

@author Jon Howell <jonh@cs.dartmouth.edu>
@classSummaryOnly true

Class **RMITest**

public class RMITest
extends java.rmi.server.UnicastRemoteObject
implements Hello

A class to test plugging SSH in under RMI using JDK1.2's SocketFactory stuff. (I had gotten this entire arrangement working once before by seriously rewiring JDK1.1.x's RMI; then they made that fix obsolete :v/ That code is in class `ssh.SecureRMIProtocol`.)

METHODS
public String `hello`( )
public static void `main`(java.lang.String []args)

Class **SecureRMIProtocol**

public class SecureRMIProtocol
extends java.lang.Object
implements SRPConstants

This class implements a simple encrypted channel, based on ssh. Several features are left out (man-in-the-middle and privacy defenses), meant to be implemented and verified by a higher layer based on the regarding calculus.

Source: based on ClientProtocol.java, my Java implementation of ssh 1.5.

@deprecated This is the version of the protocol that predates JDK1.2's RMISocketFactory mechanism. See `SSHServerSocketFactory` for the new way to plug this SSH channel implementation into RMI.

@author Jon Howell <jonh@cs.dartmouth.edu>

FIELDS
public static String `protocolVersion`

CONSTRUCTORS
public `SecureRMIProtocol`( )

METHODS
public void `accept`(java.io.InputStream is, java.io.OutputStream os)
public void `accept`(java.net.Socket socket)
public void `connect`(java.io.InputStream is, java.io.OutputStream os)
public void `connect`(java.net.Socket socket)
public void `connect`(java.lang.String host, int port)
public InputStream `getInputStream`( )
public RSAKey `getOppositeKey`( )
public OutputStream `getOutputStream`( )
public void `setKey`(ssh.RSA.RSAKey []pair)
public void `setKey`(ssh.RSA.RSAKey privateKey, ssh.RSA.RSAKey publicKey)

Class **SRPTest**

public class SRPTest
extends java.lang.Object

This simple class just tests using SSH as a raw, link protocol, without any "authentication" that the public key on the other end is meaningful. That's how we use SSH in Snowflake; Snowflake proofs take care of showing that the keys have authority.

CONSTRUCTORS
public `SRPTest`( )

METHODS
public static void `main`(java.lang.String []args)
public void `realMain`( )

Class **SSHClientSocketFactory**

public class SSHClientSocketFactory
extends java.lang.Object
implements java.rmi.server.RMIClientSocketFactory, java.io.Serializable

An adaptor to use my SSH channels with RMI from JDK 1.2, where your own socket factories can supply the channels over which RMI communicates.

CONSTRUCTORS
public `SSHClientSocketFactory`( )
public `SSHClientSocketFactory`(ssh.SSHContext context)

METHODS
public Socket `createSocket`(java.lang.String host, int port)

Class **SSHContext**

public class SSHContext
extends java.lang.Object

This class is analogous to PureTLS' SSLContext for SSL channels. It is an object that carries the state needed to connect and accept SSH channels. It has its own RSA public/private key pair, and a reference to a source of randomness.

FIELDS
public static PerThread `contextByThread`

CONSTRUCTORS
public `SSHContext`(ssh.RSA.RSAKey privateKey, ssh.RSA.RSAKey publicKey)

METHODS
public static SSHContext `getDefault`( )

Usage Get an anonymous context. Tries to use the context associated with this thread; otherwise creates a new default context.

public SDSIRSAPrivateKey `getPrivateKey`( )
public SDSIRSAPublicKey `getPublicKey`( )
public SDSIRSAPrivateKey `getSDSIRSAPrivateKey`( )

public SDSIRSAPublicKey `getSDSIRSAPublicKey`( )
public static SSHContext `newKeys`( )
public void `setKey`(ssh.RSA.RSAKey []pair)
public void `setKey`(ssh.RSA.RSAKey privateKey, ssh.RSA.RSAKey publicKey)

Class **SshInputStream**

public class SshInputStream
extends java.io.InputStream

This class reads data from an ssh stream. It extracts incoming bytes from the BinaryPacketIn packets, and buffers unused ones to return on future read requests. One way to get your hands on an instance of this class is by calling connect() and then getInputStream() on an `ssh.ClientProtocol`.

@author Jon Howell <jonh@cs.dartmouth.edu>

Constructors
public `SshInputStream`(ssh.BinaryPacketInputStream binaryIn)

Methods
public int `read`( )
public int `read`(byte []b, int off, int len)

Class **SSHOptSocket**

public class SSHOptSocket
extends java.net.Socket
implements SRPConstants

This class is a little fancier than the basic SSHSocket class in that it knows how to recognize connections back to the local VM, and optimize away the SSH handshake and encryption gunk. That saves a 1500ms public key operation (latency) and the bandwidth cost of the secret-key encryption layer. This feature is pretty important for use with RMI, which can't identify local connections on its own.

@author Jon Howell <jonh@cs.dartmouth.edu>

Constructors
public `SSHOptSocket`(ssh.SSHContext context, java.net.InetAddress remoteAddress, int remotePort)
public `SSHOptSocket`(ssh.SSHContext context, java.net.InetAddress remoteAddress, int remotePort, java.net.InetAddress localAddress, int localPort)

Usage Initiates a connection to a remote server.

public `SSHOptSocket`(ssh.SSHContext context, java.lang.String remoteHost, int remotePort)

Methods
public void `close`( )

Usage Make sure encrypted stream gets flushed cleanly.

public InetAddress `getInetAddress`( )

Usage pass through all other Socket stuff. Aaargh how I wish java.net.Socket were an interface. These stubs were automatically generated, hence the terrible parameter names.

public InputStream `getInputStream`( )
public InetAddress `getLocalAddress`( )
public int `getLocalPort`( )
public RSAKey `getOppositeKey`( )

Usage How to find out what public key identifies the other end of this connection.

public OutputStream `getOutputStream`( )
public int `getPort`( )
public synchronized int `getReceiveBufferSize`( )
public synchronized int `getSendBufferSize`( )
public int `getSoLinger`( )
public synchronized int `getSoTimeout`( )
public boolean `getTcpNoDelay`( )
public synchronized void `setReceiveBufferSize`(int p0)
public synchronized void `setSendBufferSize`(int p0)
public void `setSoLinger`(boolean p0, int p1)
public synchronized void `setSoTimeout`(int p0)
public void `setTcpNoDelay`(boolean p0)

Class **SshOutputStream**

public class SshOutputStream
extends java.io.OutputStream

This class writes data to an ssh stream. It creates a BinaryPacketOut ssh packet for each write request, and sends it down the BinaryPacketOutputStream. One way to get your hands on an instance of this class is by calling connect() and then getOutputStream() on an `ssh.ClientProtocol`.

@author Jon Howell <jonh@cs.dartmouth.edu>

Constructors

public `SshOutputStream`(ssh.BinaryPacketOutputStream binaryOut)

Methods

public void `close`( )
public void `setType`(int type)
public void `write`(byte []b, int off, int len)
public void `write`(int b)

Class **SshRandom**

public class SshRandom
extends java.util.Random

This class implements a pool of random bits, and provides access methods that are appropriate to the needs of this package.

@todo grab some actual randomness from the environment to keep the entropy flowing.
@todo or better yet, replace with a call to the new JCE.
@author Jon Howell <jonh@cs.dartmouth.edu>

CONSTRUCTORS
public `SshRandom`( )

METHODS
public BigInteger `newBigInteger`(int size)
public BigInteger `newBigIntegerBits`(int size)
public byte `newByteArray`(int size)
public int `nextByte`( )
public int `nextNonzeroByte`( )

Class **SSHServerSocket**

public class SSHServerSocket **extends** java.net.ServerSocket

An adaptor to use my SSH channels with RMI from JDK 1.2, where your own socket factories can supply the channels over which RMI communicates.

This adaptor knows how to handle SSHOptSockets and HalfSockets, as well. These classes are the implementations of SSH channel reuse and channel short-circuiting (in the local case), respectively.

CONSTRUCTORS
public `SSHServerSocket`(ssh.SSHContext context, int port)
public `SSHServerSocket`(ssh.SSHContext context, int port, int backlog, java.net.InetAddress inetaddr)

METHODS
public Socket `accept`( )

Usage Accept a connection on this socket, and run the server side of the SSH protocol on the connection to initialize it.

public void `localConnection`(java.net.Socket s)

Usage "Listen" for local connections. The caller is a local (same-VM) client who doesn't want to deal with a network connection plus a 1500ms SSH handshake overhead. He has supplied a "socket" that acts like a network socket (notably has a working InputStream and OutputStream), but is implemented entirely inside the VM. This method takes that socket and sticks it into the accept() queue, where a thread waiting to accept() on this ServerSocket will pick it up and treat it just like an incoming network connection.

Class **SSHServerSocket.AcceptThread**

public class SSHServerSocket.AcceptThread **extends** java.lang.Thread

Listen for real network connections

Constructors
public `SSHServerSocket.AcceptThread`(ssh.SSHServerSocket this$0)

Methods
public void `run`( )

Class **SSHServerSocketFactory**

public class SSHServerSocketFactory
extends java.lang.Object
implements java.rmi.server.RMIServerSocketFactory, java.io.Serializable

An adaptor to use my SSH channels with RMI from JDK 1.2, where your own socket factories can supply the channels over which RMI communicates.

Constructors
public `SSHServerSocketFactory`(ssh.SSHContext context)

Methods
public ServerSocket `createServerSocket`(int port)

Class **SSHSocket**

public class SSHSocket
extends java.net.Socket
implements KeyedSocket, SRPConstants

This class implements a simple encrypted channel, based on the ssh protocol. This class actually implements both halves; a SSHServerSocket is just a thing that accept()s requests and creates one of these SSHSockets in server mode to handle the server side of a connection.

Several features are left out (man-in-the-middle and privacy defenses); this is okay for my purposes, since I implement and verify those services in a higher layer based on my restricted-delegation logic.

Source: based on ClientProtocol.java, my Java implementation of ssh 1.5.

@author Jon Howell <jonh@cs.dartmouth.edu>

Fields
public static String `protocolVersion`

Constructors
public `SSHSocket`(ssh.SSHContext context, java.net.InetAddress remoteAddress, int remotePort)
public `SSHSocket`(ssh.SSHContext context, java.net.InetAddress remoteAddress, int remotePort, java.net.InetAddress localAddress, int localPort)

Usage Initiates a client-end connection to a remote server. ("client-end" just means that we run the client's end of the protocol.)

public `SSHSocket`(ssh.SSHContext context, java.lang.String remoteHost, int remotePort)

METHODS

public void `close`( )

Usage Make sure encrypted stream gets flushed cleanly.

public InputStream `getInputStream`( )
public RSAKey `getOppositeKey`( )

Usage How to find out what public key identifies the other end of this connection.

public OutputStream `getOutputStream`( )
public void `initClient`( )
public static void `main`(java.lang.String []args)
public static void `setBorrowingAllowed`(boolean state)
public static RSAKey `whoCalledMe`( )

Usage If you are a remote object implementation, you may call this to learn the RSAKey public key identity that authenticated the calling end of this socket. That is, in speaks-for terms, the principal returned by whoCalledMe() "says" remoteMethod(arguments...). If you call this from another method, be aware of what thread you're in. This call does its dirty work by matching the current thread with the Thread that "answered" the incoming Socket connection. So if you might be on the other side of a queue (in a different Thread) than the original RMI call, this call may return null, or worse yet, a meaningless key.

Class **StdinPasswordAuthenticator**

public class StdinPasswordAuthenticator **extends** ssh.PasswordAuthenticator

A StdinPasswordAuthenticator asks the user for his password (on stdin) when it is required. Note that Java has no provision for turning off echoing of characters on stdin.

Source: for ssh protocol definition: draft-ylonen-ssh-protocol-00.txt especially pages 13, 21.

@author Jon Howell <jonh@cs.dartmouth.edu>

CONSTRUCTORS

public `StdinPasswordAuthenticator`( )

METHODS

public void `authenticate`(ssh.BinaryPacketInputStream binaryIn, ssh.BinaryPacketOutputStream binaryOut)

Class **StreamExtras**

public class StreamExtras **extends** java.lang.Object

Tools used to work with BigIntegers on DataInput and DataOutput streams.

CONSTRUCTORS

public `StreamExtras`( )

METHODS

public static BigInteger `readBigInteger`(java.io.DataInput dis)

Usage Reads a BigInteger from the stream in the form specified by the ssh 1.5 internet-draft.

public static void `writeBigInteger`(java.io.DataOutput dos, java.math.BigInteger bi)

Usage Writes a BigInteger to the stream in the form specified by the ssh 1.5 internet-draft.

Class **Terminal**

public class Terminal
extends java.lang.Object

Use my ssh protocol to connect to an sshd server.

Warning: this class does no checking of the authenticity of the remote host's public key.

It does allow the user to authenticate to the host using a password, of course, since the host will not allow the connection without some form of user authentication. **Warning:** the password is *echoed to the console*.

The class spawns two threads that

- read data from System.in and write it to an OutputStream, and
- read data from some InputStream and write it to System.out.

Another thread wakes up periodically to do nothing, which works around a bug in System.in, to keep it from blocking all the other threads.

BUGS: System.in can't read in increments smaller than a line.

@author Jon Howell <jonh@cs.dartmouth.edu>

CONSTRUCTORS

public `Terminal`(java.io.InputStream in, java.io.OutputStream out)

Package ssh.RSA

This package is my own implementation of RSA encryption for my `ssh` class. It is factored out neatly because I figured something like the Java Cryptography Extension (JCE) would come along to replace it.

@todo Replace with JCE calls. Use cryptix, or turn this package into a JCE provider.
@author jonh@cs.dartmouth.edu

Classes

Class **Keygen**

public class Keygen
extends java.lang.Object

RSA.Keygen: generates new RSA keys.

Source: modeled after ssh-1.2.22/rsa.c:rsa_generate_key()
@author Jon Howell jonh@cs.dartmouth.edu

METHODS
public static RSAKey `generateKeys`(int bits, ssh.SshRandom random)
public static RSAKey `generateKeys`(ssh.SshRandom random)

Class **main**

public class main
extends java.lang.Object

A Unix command-line interface for creating new RSA key pairs.

CONSTRUCTORS
public `main`( )

METHODS
public static void `main`(java.lang.String []args)

Class **RSAKey**

public class RSAKey
extends java.lang.Object
implements java.io.Serializable, java.security.interfaces.RSAPublicKey, java.security.interfaces.RSAPrivateKey

The RSAKey class holds half of an RSA key pair, and performs RSA encryption and decryption (or signing and verifying) operations.
@author Jon Howell <jonh@cs.dartmouth.edu>

SERIALIZABLE FIELDS
public int `bits`
public BigInteger `exponent`
public BigInteger `modulus`

CONSTRUCTORS
public `RSAKey`( )

METHODS
public BigInteger `cryptBasic`(java.math.BigInteger input)
public byte `decrypt`(java.math.BigInteger input)
public BigInteger `encrypt`(byte []from, int fromoff, int fromlen, ssh.SshRandom random)
public BigInteger `encrypt`(byte []from, ssh.SshRandom random)

public boolean `equals`(java.lang.Object o)
public static RSAKey `fromRSAPrivateKey`(java.security.interfaces.RSAPrivateKey pk)
public static RSAKey `fromRSAPublicKey`(java.security.interfaces.RSAPublicKey pk)
public String `getAlgorithm`( )
public byte `getEncoded`( )
public String `getFormat`( )
public BigInteger `getModulus`( )
public BigInteger `getPrivateExponent`( )
public BigInteger `getPublicExponent`( )
public int `hashCode`( )
public static RSAKey `nullKey`( )
public BigInteger `randomPad`(byte []from, int fromoff, int fromlen, ssh.SshRandom random)
public void `readAsciiSsh`(java.io.InputStream is)
public static RSAKey `readSerialized`(java.io.DataInput di)
public static RSAKey `readSsh`(java.io.DataInput di)
public byte `toByteArray`( )
public byte `unpad`(java.math.BigInteger input)
public void `writeSerialized`(java.io.DataOutput dop)
public void `writeSsh`(java.io.DataOutput dop)

Class **test**

public class test **extends** java.lang.Object

Routines to test my RSAKey implementation.

CONSTRUCTORS
public `test`( )

METHODS
public static void `main`(java.lang.String []args)

Package ssl

This package consists of wiring to attach the PureTLS implementation of SSL/TLS to RMI. (See `http://www.rtfm.com/puretls/`.)

I never got it working robustly; it always had these mysterious long delays. Hence I stuck with my `ssh` implementation instead. I abandoned this class before it got far enough to have support for actual Snowflake protocol; all it does right now is route RMI messages over SSL channels.

Classes

Class **SfContext**

public class SfContext
extends COM.claymoresystems.ptls.SSLContext

An SSLContext with some handier functions for Snowflake use, such as for setting the private/public keypair.

Constructors
public `SfContext`( )

Methods
public void `setPrivateKey`(java.security.PrivateKey pk)

Class **SSLClientSocketFactory**

public class SSLClientSocketFactory
extends java.lang.Object
implements java.rmi.server.RMIClientSocketFactory, java.io.Serializable

A SSLClientSocketFactory lives on the client side of the RMI connections, and creates SSL connections back to the server to transmit RMI messages.

Constructors
public `SSLClientSocketFactory`( )
public `SSLClientSocketFactory`(COM.claymoresystems.ptls.SSLContext context)

Methods
public Socket `createSocket`(java.lang.String host, int port)

Class **SSLServerSocketFactory**

public class SSLServerSocketFactory
extends java.lang.Object
implements java.rmi.server.RMIServerSocketFactory, java.io.Serializable

A factory to create RMI server (listener) sockets. Remote objects specify this factory class when they invoke UnicastRemoteObject's constructor to demand that clients connect to the object via an SSL channel.

Constructors
public `SSLServerSocketFactory`(COM.claymoresystems.ptls.SSLContext context)

Methods
public ServerSocket `createServerSocket`(int port)

Package timingexp

This package includes tools for timing parts of Snowflake, both for diagnostic and evaluative purposes. The primary class for evaluation is `GenerateTestCases`, which drives various series of tests of the Snowflake versions of HTTP and RMI.

Interfaces

Interface **NullRMICall**

public interface NullRMICall **extends** java.rmi.Remote

Used with TestJavaOverheads.

METHODS
public Object `nullMethod`( )

Interface **TestRMICall**

public interface TestRMICall **extends** java.rmi.Remote

Used with RMIExp.

METHODS
public Object `requestFile`(java.lang.String path)

Interface **TestRMIReconfigureIfc**

public interface TestRMIReconfigureIfc **extends** java.rmi.Remote

An interface for reconfiguring the TestRMIServer between RMIExp experiments.

METHODS
public void `setCacheNotVeryUseful`(boolean state)

Classes

Class **Experiment**

public abstract class Experiment **extends** java.lang.Object

The two classes of experiments, RMIExp and HTTPExp, extend this abstract class. GenerateTestCases uses this class as a generic way to invoke either kind of experiment.

CONSTRUCTORS
public `Experiment`( )

Class **GenerateTestCases**

public class GenerateTestCases **extends** java.lang.Object

GenerateTestCases is the master test harness for the timings in the Measurement chapter of my thesis. It produces all permutations of variables in several dimensions, then kicks out the cases we can't or don't want to test (not applicable, unimplemented, or too slow). The resulting list is indexed by "test case number," so we can specify an integer to skip over some preceding number of tests.

The Overview for this manual tells how to reproduce specific experiments from the dissertation.

Constructors
public `GenerateTestCases`( )

Methods
public void `axis`(java.lang.String optName, int []values)
public void `axis`(java.lang.String optName, java.lang.Object value)
public void `axis`(java.lang.String optName, java.lang.Object []values)
public void `dmain`(java.lang.String []args)
public static void `main`(java.lang.String []args)

Class **HttpExp**

public class HttpExp
extends timingexp.Experiment

A class of experiments that measure the relative speeds of various types of HTTP requests.

Fields
public static Timeline `timeline`

Methods
public static void `main`(java.lang.String []args)
public Options `optionsFactory`( )

Usage Create a default Options object, which the GenerateTestCases harness will populate with the actual options.

public void `runExperiment`(Tools.Options opts)

Class **NullRMICallImpl**

public class NullRMICallImpl
extends java.rmi.server.UnicastRemoteObject
implements NullRMICall

Used with TestJavaOverheads

Constructors
public `NullRMICallImpl`( )

Methods
public Object `nullMethod`( )

Class **RMIExp**

public class RMIExp
extends timingexp.Experiment

A class of experiments to measure the performance of RMI, RMI/ssh, RMI/Sf, maybe later RMI/SSL.

Fields

public static Timeline `timeline`

METHODS

public void `flushChannels`( )

public static void `main`(java.lang.String []args)

public Options `optionsFactory`( )

Usage Create a default Options object, which the GenerateTestCases harness will populate with the actual options.

public void `runExperiment`(Tools.Options opts)

Class **TestJavaOverheads**

public class TestJavaOverheads **extends** java.lang.Object

Get some rough estimates on overheads of Java for "performance of name resolution" thesis section (2.5.4)

METHODS

public static void `main`(java.lang.String []args)

public void `realMain`( )

Class **TestResult**

public class TestResult **extends** java.lang.Object **implements** java.io.Serializable

A debugging class used to ensure that the Test RMI interface was really transmitting the bytes it claimed to be transmitting.

CONSTRUCTORS

public `TestResult`(byte []buf)

METHODS

public static int `cheesyChecksum`(byte []b)

public void `verify`( )

Class **TestRMICallBasic**

public class TestRMICallBasic **extends** timingexp.TestRMICallImpl

Just a basic RMI connection; no authorization framework. Used with RMIExp.

CONSTRUCTORS

public `TestRMICallBasic`( )

Usage Create an RMI-only object.

METHODS

protected void `checkAuth`( )

Class **TestRMICallImpl**

public abstract class TestRMICallImpl
extends java.rmi.server.UnicastRemoteObject
implements TestRMICall

A Snowflake-authorized version of TestRMICall, used with RMIExp.

The ssh/Snowflake code in here was basically stripped from relational.InternalDatabase.

CONSTRUCTORS

protected `TestRMICallImpl`( )
protected `TestRMICallImpl`(int port, java.rmi.server.RMIClientSocketFactory csf, java.rmi.server.RMIServerSocketFactory ssf)

METHODS

public Object `requestFile`(java.lang.String path)

Class **TestRMICallSf**

public class TestRMICallSf
extends timingexp.TestRMICallSsh

RMI over Snowflake.

CONSTRUCTORS

public `TestRMICallSf`(ssh.SSHContext context, sdsi.SDSIPrincipal serverIssuer)

Usage Create an RMI-over-Snowflake object.

METHODS

public Object `requestFile`(java.lang.String path)

Class **TestRMICallSsh**

public class TestRMICallSsh
extends timingexp.TestRMICallImpl

RMI over ssh.

CONSTRUCTORS

public `TestRMICallSsh`(ssh.SSHContext context)

Usage Create an RMI-over-SSH object.

Class **TestRMICallSsl**

public class TestRMICallSsl
extends timingexp.TestRMICallImpl

RMI over SSL.

CONSTRUCTORS

public `TestRMICallSsl`(COM.claymoresystems.ptls.SSLContext context)

Usage Create an RMI-over-SSL object.

Class **TestRMIReconfigure**

public class TestRMIReconfigure **extends** java.rmi.server.UnicastRemoteObject **implements** TestRMIReconfigureIfc

A remote object for reconfiguring the TestRMIServer between RMIExp experiments.

Constructors
public `TestRMIReconfigure`( )

Methods
public void `setCacheNotVeryUseful`(boolean state)

Class **TestRMIServer**

public class TestRMIServer **extends** java.lang.Object

The server side of the RMI performance test harness. Sets up one object of each type to be served; analogous to the fourServers mode of SecureServerConfig.

Methods
public static void `main`(java.lang.String []args)

Class **TheRace**

public class TheRace **extends** java.lang.Object

A race between base64'ing a canonical sexp and to/from readable strings. I was trying to decide whether it was cheaper to transmit canonical S-expressions in Base64, or to use the advanced S-expression encoding. It turned out not to matter much; the overhead is dominated by the generally abysmal S-expression parsing code.

Constructors
public `TheRace`( )

Methods
public static void `main`(java.lang.String []args)
public void `realMain`(java.lang.String []args)

Class **Timeline**

public class Timeline **extends** java.lang.Object

A debugging tool. Used to look for big delays in a code path, delays on the order of several milliseconds. Amazing how many such huge delays there are floating around in the `sdsi` package and elsewhere.

When called from the command line, run all three processes (proxy, mail servlet, secure-database) in a single process, and let them all demarcate times on the same timeline. Of course, this'll screw things up since some ssh/RMI will be optimized away...

Constructors

public `Timeline`( )

METHODS

public static NumberFormat `getNF`( )
public static void `main`(java.lang.String []argv)
public static void `timePoint`(java.lang.String desc)
public static void `zeroTimer`( )

Package Tools

A collection of miscellaneous tools that do not belong in any other package. Many are debugging tools.

Interfaces

Interface **BinarySearch.Test**

public static interface BinarySearch.Test

Users of BinarySearch must implement the Test class to report the true/false values for any given integer.

Methods

public boolean `test`(int value)

Usage return false for smaller values and true for values larger than or equal to the desired value.

Classes

Class **Arrays**

public class Arrays **extends** java.lang.Object

Tools to manipulate arrays. Some of these are obsolete with the introduction of `java.util.Arrays` in JDK 1.2.

Constructors

public `Arrays`( )

Methods

public static String `dumpBytes`(byte []bytes)
public static String `dumpBytes`(byte []bytes, int off, int len)
public static boolean `equals`(byte []a, byte []b)
public static void `setByteArray`(byte []a, byte val, int start, int length)
public static void `setIntArray`(int []a, int val, int start, int length)
public static void `zeroByteArray`(byte []a)
public static void `zeroByteArray`(byte []a, int start, int length)
public static void `zeroIntArray`(int []a)
public static void `zeroIntArray`(int []a, int start, int length)

Class **Assert**

public class Assert **extends** java.lang.Object

A simple assertion-checking call that throws a RuntimeException if the check fails. Java provides no way to "compile these out," so you'll always be doing whatever work you do to generate the boolean condition you're testing. But by using a consistent method call, you can later mechanically remove the checks for performance. Using this method is just a way of indicating "this is an optional test to make debugging easier."

Constructors

public `Assert`( )

METHODS
public static void `assert`(boolean premise)
public static void `assert`(boolean premise, java.lang.String s)
public static boolean `getEnabled`( )

Class **BinarySearch**

public class BinarySearch
extends java.lang.Object

Perform a binary search over bounded or unbounded integer intervals.

CONSTRUCTORS
public `BinarySearch`( )

METHODS
public static int `interval`(int min, int max, Tools.BinarySearch.Test test)

Usage Search a bounded interval. The interval includes `min` and `max`.

public static void `main`(java.lang.String []args)
public static int `unbounded`(int bound, boolean searchAbove, Tools.BinarySearch.Test test)

Parameters
- `bound` - The known bound
- `searchAbove` - When true, `bound` represents the min bound of the search space.
- `test` - The closure object that knows the truth value at any given int.
  test.test(x)==false && test.test(y)==true should always imply x<y.

Class **ByteBuffer**

public class ByteBuffer
extends java.lang.Object

The byte[] analog of StringBuffer. Lets a tree of data structures recursively generate a linearized representation of the data structure without repeated byte[] allocations, copies, and deletions. This code actually does reallocate and copy the byte[], of course, but only log n times, and starting with a pretty big buffer.

@author jonh@cs.dartmouth.edu

CONSTRUCTORS
public `ByteBuffer`( )
public `ByteBuffer`(int initialAllocation)

METHODS
public void `append`(byte b)

Usage Append a single byte to the end of the buffer.

public void `append`(byte []inBuf)

Usage Append a byte array to the end of the buffer.

public void `append`(byte []inBuf, int inOff, int inLen)

Usage Append part of a byte array to the end of the buffer.

public boolean `equals`(java.lang.Object o)
public byte `getRawBytes`( )

Usage These three methods let you get at the byte array itself without making a data copy. Useful, for example, if you just want to write() it directly to a socket. Note that getRawBytes() has reference semantics: if you dink around with the returned buffer, you'll change the contents of this ByteBuffer object. TODO: Perhaps I should make these methods 'protected' in this class, and create a subclass RawByteBuffer that exposes them?

public int `getRawOffset`( )
public int `hashCode`( )
public int `length`( )
protected void `reallocate`(int minLength)

Usage Reallocate the internal buffer. Invariant: when this call returns, buf.length >= minLength, and off hasn't changed.

public byte `toByteArray`( )

Usage Returns a new byte[] containing the contents of this buffer, trimmed to length.

public byte `toByteArray`(byte []outBuf, int outOff)

Usage Copies this.length() bytes into outBuf starting at outOff.

Class **ChainInputStream**

public class ChainInputStream **extends** java.io.InputStream

Builds a "longer" InputStream out of two others. When the first input stream runs out, read() requests will be satisfied from the second input stream. Supports mark() and reset(), even across stream boundaries, when both input streams support mark and reset.

CONSTRUCTORS

public `ChainInputStream`(java.io.InputStream s1, java.io.InputStream s2)

METHODS

public int `available`( )
public void `close`( )
public synchronized void `mark`(int readlimit)
public boolean `markSupported`( )
public int `read`( )
public int `read`(byte []buf)
public int `read`(byte []buf, int offset, int length)
public synchronized void `reset`( )
public long `skip`(long n)

Class **CopyStream**

public class CopyStream
extends java.lang.Object

Copy an InputStream to an OutputStream. This while() loop idiom seems to turn up enough that it belongs in a Tools method.

Constructors
public `CopyStream`( )

Methods
public static void `copyStream`(java.io.InputStream is, java.io.OutputStream os)
public static void `copyStream`(java.io.InputStream is, java.io.OutputStream os, int bufSize)

Class **CountingFilterOutputStream**

public class CountingFilterOutputStream
extends java.io.FilterOutputStream

An OutputStreamFilter that counts the number of bytes written.

Constructors
public `CountingFilterOutputStream`(java.io.OutputStream out)

Methods
public long `getCount`( )
public void `write`(byte []b)
public void `write`(byte []b, int off, int len)
public void `write`(int b)

Class **CRC32**

public class CRC32
extends java.lang.Object
implements java.util.zip.Checksum

Computes 32-bit Cyclic Redundancy Checks. Allows the use of arbitrary polynomials.

Source: The core CRC computation (update()) and the buildTable routine are based on those in a public-domain Pascal program called "CRC_Calc" by F. Martin Richardson, Jr. I found his code at http://www.csd.net/ cgadd/knowbase/CRC0019.HTM Excerpted comments from his program: Routines for calculations derived with the help of Doctor Dobb's Journal #188, MAY 1992. ... This file is hereby commited to the public domain. Feel free to use it in your development. All I ask is a little recognition if you use it in your software. Also, if this file is modified in any way and re-distributed, please retain the credits to the people who wrote the routines.

@author Jon Howell <jonh@cs.dartmouth.edu>

Constructors
public `CRC32`( )

public `CRC32`(int polynomial)

METHODS
public static long `crc32`(byte []s)
public long `getValue`( )
public static void `main`(java.lang.String []args)
public void `reset`( )
public void `update`(byte []s)
public void `update`(byte []s, int off, int len)
public void `update`(int b)

Class **DeadManSwitch**

public class DeadManSwitch **extends** java.lang.Thread

When profiling, the program must exit without a signal. But if you're trying to profile an RMI call, you can't System.exit() before you return() your results, or you miss the reply time. So this class lets you set a timer, then exit after the return call has completed.

METHODS
public void `run`( )
public static void `setTimer`(long millis)

Class **DumpProf**

public class DumpProf **extends** java.lang.Object

Send self SIGQUIT to cause -prof info to get dumped to output file. Not sure how to get it to reset, too; that would be really nice. Better still would be a Java interface to the hprof module.

More in jdk1_2_2-src/src/share/tools/hprof.

CONSTRUCTORS
public `DumpProf`( )

METHODS
public static native void `dump`( )

Class **Endian**

public class Endian **extends** java.lang.Object

Tools to transfer multibyte data into and out of byte[] arrays, with either endianness.

CONSTRUCTORS
public `Endian`( )

METHODS
public static int `BigGetInt`(byte []b, int off)
public static long `BigGetLong`(byte []b, int off)

public static short `BigGetShort`(byte []b, int off)
public static void `BigPutInt`(byte []b, int off, int value)
public static void `BigPutLong`(byte []b, int off, long value)
public static void `BigPutShort`(byte []b, int off, short value)
public static int `LittleGetInt`(byte []b, int off)
public static long `LittleGetLong`(byte []b, int off)
public static short `LittleGetShort`(byte []b, int off)
public static void `LittlePutInt`(byte []b, int off, int value)
public static void `LittlePutLong`(byte []b, int off, long value)
public static void `LittlePutShort`(byte []b, int off, short value)

Class **HashKey**

public class HashKey **extends** java.util.Vector

A class used to hash uniquely on a combination of inputs. Two HashKeys are equal (and have the same hashcode) when the same is true of all of their members, pairwise.

CONSTRUCTORS
public `HashKey`( )

METHODS
public boolean `equals`(java.lang.Object o)
public int `hashCode`( )

Class **Hex**

public class Hex **extends** java.lang.Object

Tools for manipulating hexadecimal strings.

CONSTRUCTORS
public `Hex`( )

METHODS
public static byte `bytesToHex`(byte []binary)
public static byte `hexToBytes`(byte []hex)

Class **IndentWriter**

public class IndentWriter **extends** java.io.FilterWriter

A FilterWriter stream that inserts space after each linefeed to indent its output.
@author jonh@cs.dartmouth.edu

CONSTRUCTORS
public `IndentWriter`(java.io.Writer out)

METHODS
public void `addIndent`(int increment)
Usage Adjust the number of spaces to indent by some (positive or negative) increment.

public void `flush`( )
public static void `main`(java.lang.String []args)
public void `print`(java.lang.String s)
public void `println`( )
public void `println`(java.lang.String s)
public void `setIndent`(int depth)

Usage Set the number of spaces to indent to an absolute value.

public void `write`(char []cbuf, int off, int len)
public void `write`(int c)
public void `write`(java.lang.String s)
public void `write`(java.lang.String str, int off, int len)
protected void `writeIndent`( )

Usage Output `depth` spaces to indent a line.

protected void `writeLine`(java.lang.String s, int off, int len)

Class **Log**

public class Log **extends** java.lang.Object

Tools for logging messages to the console (or another OutputStream). Messages can belong to different categories (`level`s), analogous to syslog.

CONSTRUCTORS
public `Log`( )
public `Log`(java.io.OutputStream os)
public `Log`(java.io.OutputStream os, java.lang.String prefix)
public `Log`(java.lang.String prefix)

METHODS
public Log `addLevel`(java.lang.String level)
public void `log`(java.lang.String message)
public void `log`(java.lang.String level, java.lang.String message)
public void `logc`(java.lang.String level, java.lang.String message)
public OutputStream `logs`(java.lang.String level)
public boolean `logt`(java.lang.String level)
public PrintWriter `logw`(java.lang.String level)
public Log `setPrefix`(java.lang.String prefix)

Class **LRUHashMap**

public class LRUHashMap **extends** java.util.HashMap

A HashMap that roughly bounds the size of storage consumed, and kicks out keys whenever they haven't been accessed in a long time.

@todo A clock algorithm (as in OS buffer cache pages) might be faster in some situations.

CONSTRUCTORS
public `LRUHashMap`( )
public `LRUHashMap`(int initialCapacity, int maxOccupancy, float loadFactor)

Parameters

`maxOccupancy` - - how many keys can live in the LRUHashMap at once. Eventually this might be in terms of a size() parameter called on the keys.

METHODS
public Object `clone`( )
public Object `get`(java.lang.Object key)
public Object `put`(java.lang.Object key, java.lang.Object value)

Usage Sometimes put() will eject other key(s) from the hashtable. (at most one key until I implement a notion of per-entry size) Note that this class currently does not allow you to hash null values.

protected void `reap`( )
public Object `remove`(java.lang.Object key)
public Collection `values`( )

Class **MakeDebugClass**

public class MakeDebugClass
extends java.lang.Object

Reads a class' definition using reflection, and spits out a subclass (or implementation) that adds debugging comments before each method call. Useful for all sorts of mechanically-generated tweaks to existing classes.

CONSTRUCTORS
public `MakeDebugClass`( )

METHODS
public static String `javaName`(java.lang.Class c)
public static void `main`(java.lang.String []args)
public void `realMain`(java.lang.String []args)

Class **MD5**

public class MD5
extends java.lang.Object

This class computes MD5 hashes.

Introduction: To compute the message digest of a chunk of bytes, create an MD5 object "md5", call md5.update() as needed on buffers full of bytes, and then call md5.getValue(), which will fill a supplied 16-byte array with the digest.

A main() method is included that hashes the data on System.in.

It seems to run around 25-30 times slower (JDK1.1.6) than optimized C (gcc -O4, version

2.7.2.3). Measured on a Sun Ultra 5 (SPARC 270MHz).

SOURCE: Manually translated from some public domain C code (md5.c) included with the ssh-1.2.22 source. Comments from ssh-1.2.22/md5.c, the basis for this code:

This code has been heavily hacked by Tatu Ylonen <ylo@cs.hut.fi> to make it compile on machines like Cray that don't have a 32 bit integer type.

This code implements the MD5 message-digest algorithm. The algorithm is due to Ron Rivest. This code was written by Colin Plumb in 1993, no copyright is claimed. This code is in the public domain; do with it what you wish.

Equivalent code is available from RSA Data Security, Inc. This code has been tested against that, and is equivalent, except that you don't need to include two pages of legalese with every copy.

To compute the message digest of a chunk of bytes, declare an MD5Context structure, pass it to MD5Init, call MD5Update as needed on buffers full of bytes, and then call MD5Final, which will fill a supplied 16-byte array with the digest.
@deprecated JCE now includes an interface for computing message digests.
@author Jon Howell <jonh@cs.dartmouth.edu>

CONSTRUCTORS
public `MD5`( )

METHODS
public byte `getValue`( )
public void `getValue`(byte []digest)
public int `hashCode`( )
Usage The hashCode of an MD5 hash is useful in that it doesn't reveal any more information about the original hashed object that does MD5. It's not as useful for its uniqueness, since it's only 32 bits long and not 128.

public static void `main`(java.lang.String []args)
public void `update`(byte []buf)
public void `update`(byte []newbuf, int bufstart, int buflen)
public void `update`(int b)

Class **Memory**

public class Memory **extends** java.lang.Object

A Tool for examining the current amount of memory in use by the JVM. Great for finding memory "leaks" such as leaving things in hash tables that you didn't intend. I first used it to get a handle on how much memory I was using when indexing the cells in my `relational` databases.

CONSTRUCTORS

public `Memory`( )

METHODS
public double `getMB`( )
public long `getMemory`( )
public void `lap`( )
public String `toString`( )

Class **Mnemonic**

public class Mnemonic
extends java.lang.Object

A class to give a wordy mnemonic to an otherwise meaningless bitstring (say, a hash value). Returns a string made up of two four-letter words (not like that!), such as "duke-alma." There are 1024 words in this class' vocabulary, so a two-word string is a mnemonic with 20 bits of uniqueness.

CONSTRUCTORS
public `Mnemonic`( )

METHODS
public static String `get`(long word)
public static String `get20`(byte []bs)

Class **MultiMap**

public class MultiMap
extends java.lang.Object

A Map whose members are sets.

CONSTRUCTORS
public `MultiMap`( )

METHODS
public void `add`(java.lang.Object from, java.lang.Object to)

Usage The nicest operation of this class: Add object `to` to the set of objects identified by the key `from`.

public Set `getSet`(java.lang.Object from)

Usage Get the set of objects associated with the key `from`.

public Iterator `iterator`(java.lang.Object from)
public Set `keys`( )
public int `size`( )
public int `size`(java.lang.Object from)

Class **NullOutputStream**

public class NullOutputStream
extends java.io.OutputStream

An OutputStream that discards all writes. Boy, that sure was easy to code!

CONSTRUCTORS

public `NullOutputStream`( )

METHODS

public void `write`(byte []b)

public void `write`(byte []b, int off, int len)

public void `write`(int b)

Class **Options**

public class Options **extends** java.lang.Object

A tool for parsing options from the command line. Nice because the same definition is used to determine how options are parsed *and* generate a helpful usage message. Can be subclassed to declare the option types.

FIELDS

public boolean `allowExtraOptions`

- Does the program allow the user to "make up" option names?

  Subclass may set this in defineOptions().

public boolean `allowExtraArguments`

- Does the program allow the user to append extra unnamed arguments?

  Subclass may set this in defineOptions().

public String `programName`

- What is the program called, so usage() prints something meaningful?

  Subclass may set this in defineOptions().

CONSTRUCTORS

public `Options`( )

public `Options`(java.lang.String []argParam)

METHODS

public void `defineArgument`(java.lang.String argName, boolean required, java.lang.String description, java.lang.String defaultValue)

Usage Define an argument (an input without an -optName tag)

public void `defineOption`(java.lang.String optName, java.lang.String description, java.lang.String defaultValue)

Usage Define an option, a string (argVal) on the command line preceded by "-argName=argVal" or "-argName argVal".

public void `defineOptions`( )

Usage Subclasses override this method to define the set of acceptable options and arguments. This method should call `defineOption()`, `defineArgument()` to set up the definitions. Set `allowExtraOptions` and `allowExtraArguments` to control whether extra options and arguments (beyond those defined) are allowed. Set `programName` to determine how the program name appears in the usage() display.

public void `defineProgram`(java.lang.String programName)

Usage Define the name of this program, as it appears in the usage() display.

public void `dumpOptions`( )
public void `dumpOptions`(java.io.OutputStream out)
public String `get`(java.lang.String name)

Usage Get an argument or option by name

public String `get`(java.lang.String name, java.lang.String defaultValue)

Usage Get an argument or option by name, supplying a dynamic default value. Useful when an argument or option has a default value that isn't a static string, but can be computed by the time the argument is fetched.

public String `getArg`(int index)

Usage Get an argument by position (0-indexed). Useful if you're being lazy and using this class directly, rather than subclassing it to give your arguments names.

public boolean `getBoolean`(java.lang.String name)
public int `getInt`(java.lang.String name)
public void `optionError`(java.lang.String error)

Usage override this method to do something different that barf on stderr if options don't parse out correctly. When you're done, throw Options.OptionException to cause option parsing to stop.

public String `pad`(java.lang.String arg, int len)
public void `setOption`(java.lang.String optName, java.lang.String optValue)
public void `suppressUsage`( )
public void `usage`( )

Usage Outputs usage info to stderr.

Class **Options.OptEntry**

public class Options.OptEntry **extends** java.lang.Object

Describes an option or an argument.

Class **Perly**

public class Perly **extends** java.lang.Object

A sort routine and a way to get the list of keys from a Hashtable as a Vector (rather than a nasty Enumeration).

CONSTRUCTORS
public `Perly`( )

METHODS
public static Vector `keys`(java.util.Hashtable hash)
public static Vector `sort`(java.util.Vector list)

Class **PerThread**

public class PerThread **extends** java.lang.Object

A tool to organize data that should be stored per-thread. Cannot inherit bindings when creating subthreads.

@todo I think Java has finally added support for this sort of thing in JDK 1.2 — look into it and deprecate this class.

CONSTRUCTORS
public `PerThread`( )
public `PerThread`(java.lang.Object defaultObject)

METHODS
public Object `get`( )

Usage Return the object associated with this thread, or the default object if there is none.

public void `set`(java.lang.Object object)

Usage Establish the object to associate with the current thread.

public void `setDefault`(java.lang.Object defaultObject)

Usage Establish which object should be returned on a get() call if no object is yet defined for the calling thread

Class **PrefixMap**

public class PrefixMap **extends** java.lang.Object

A data structure kind of like a Map, except that the get() operation returns the stored value associated with the longest key that's a prefix of the argument key.

I use it to look up information bound to a URL or any prefix of it in `jp.SfUserAgent`.

CONSTRUCTORS
public `PrefixMap`( )

Methods
public Object `get`(java.lang.String key)
public void `put`(java.lang.String key, java.lang.Object value)

Class **ProgressBar**

public class ProgressBar
extends java.lang.Object

Display an ASCII progress bar to satiate users during slow operations.

Constructors
public `ProgressBar`(int range)

Methods
public void `done`( )
protected String `update`(double frac)
public void `update`(int value)

Class **Props**

public class Props
extends java.lang.Object

Print out the system-defined list of Properties.

Constructors
public `Props`( )

Methods
public static void `main`(java.lang.String []args)

Class **RecordingInputStream**

public class RecordingInputStream
extends java.io.FilterInputStream

A class that nonintrusively records some section of an input stream as it is read, so that it may be rewound and "reviewed" later. This is less "invasive" than using a BufferedInputStream, which although it can be rewound, will "oversuck" the underlying stream, so that when you're done with the underlying stream, it's left in an undetermined state.

Constructors
public `RecordingInputStream`(java.io.InputStream is)

Methods
public byte `getRecordingAsBytes`( )
public InputStream `getRecordingAsStream`( )
public int `read`( )
public int `read`(byte []p0)
public int `read`(byte []p0, int p1, int p2)
public long `skip`(long p0)

Usage Calling skip() while recording has an undefined impact on the recording. It might record the skipped bytes, or omit them from the recording, or make a recording of YMCA by the Village People.

public void `startRecording`( )

Usage Start a new recording (clearing any previous recording).

public void `stopRecording`( )

Class **SmallHashset**

public class SmallHashset **extends** java.lang.Object

Used as a (not drop-in) replacement for HashSet (or Hashtable with meaningless values attached to keys); saves oodles of memory. HashSet uses about 40 bytes per key – 16 for the Entry structure {hash, key, value, next}, 4 for its Class pointer (guessing), probably 8 for its mallocing (length, next fields?)... the 40 value was empirically measured, and includes the unused bucket head slots in the table (25-62%).

Anyway, so we beat that by trading time for space. Each occupied entry takes only 4 bytes (Object pointer). However, we might spend more time hopping down the array (HashSet only has to hop down the linked list corresponding to one bucket, which probabilistically will stay low due to the count/threshold mechanism), and we don't save the hashCode() of the keys, so we might spend more time recomputing those on rehashes. This SmallHashset uses (empirically) about 8 bytes/entry, accounting for the 25-62% empty table slots.

Increasing loadFactor (bad idea) or decreasing the factor of 2 used when rehashing will reduce the space requirements further. However, a higher loadFactor increases (rapidly) the average time spent walking the table, and a lower growth rate increases the time spent rehashing.

Since my application (indexing) may often involve growing a Hashtable and then using it read-only, it may pay to add a method to rehash one final time to make a fairly tight fit.

Constructors

public `SmallHashset`( )
public `SmallHashset`(int capacity)

Methods

public boolean `containsKey`(java.lang.Object key)
protected boolean `containsKey`(java.lang.Object key, int index)
public Iterator `getKeyIterator`( )
public int `indexFor`(java.lang.Object key)
protected void `internalPut`(java.lang.Object key, int index)
public void `put`(java.lang.Object key)
public int `size`( )

Class **SnoopyIn**

public class SnoopyIn **extends** java.io.FilterInputStream

Insert this filter among your `InputStream`s to have the data passing through the stream logged with the `Log` tool.

@classConcise true

Constructors

public `SnoopyIn`(java.io.InputStream in, Tools.Log log)

Class **SnoopyOut**

public class SnoopyOut
extends java.io.FilterOutputStream

Insert this filter among your `OutputStream`s to have the data passing through the stream logged with the `Log` tool.

@classConcise true

Constructors

public `SnoopyOut`(java.io.OutputStream out, Tools.Log log)

Methods

public void `comment`(java.lang.String s)

Usage Send an explicit message to the log.

Class **TeeOutputStream**

public class TeeOutputStream
extends java.io.FilterOutputStream

A TeeOutputStream makes a copy of every write onto another stream on the side.

Note: the primary stream is given first shot at the write; if it throws an exception, the secondary stream doesn't see the write.

Constructors

public `TeeOutputStream`(java.io.OutputStream primary, java.io.OutputStream secondary)

Methods

public void `close`( )
public void `flush`( )
public void `write`(byte []b)
public void `write`(byte []b, int off, int len)
public void `write`(int b)

Class **Text**

public class Text
extends java.lang.Object

Yet another tool for indenting text strings.

Constructors

public `Text`( )

METHODS

public static String `indent`(int d)
public static String `indent`(int d, java.lang.String s)
protected static String `realIndent`(int d)

Class **ThreadTool**

public class ThreadTool
extends java.lang.Object

A debugging tool for figuring out which code is running in which thread.

CONSTRUCTORS

public `ThreadTool`( )

METHODS

public static void `threadInfo`( )

Class **Timer**

public class Timer
extends java.lang.Object

A tool for inspecting the current process and wall-clock times. I use it both for analyzing slow code and for running the experiments in the `timingexp` package.

FIELDS

public static int `utimeOff`
public static int `stimeOff`
public static int `cutimeOff`
public static int `cstimeOff`
public static int `clkTckOff`
public static int `wallSec`
public static int `wallNsec`

CONSTRUCTORS

public `Timer`( )

METHODS

public float `cstime`( )
public float `cutime`( )
public static NumberFormat `getNF`( )
public float `getTime`(int i)
public void `lap`( )
public void `reset`( )
public float `stime`( )
public static native long `syscallTimes`( )
public String `toString`( )
public void `unlap`( )

Usage Use this method after reading out the time to continue timing from the same start point.

public float `utime`( )
public double `wallTime`( )

Exceptions

Interface **Options.OptionException**

public static class Options.OptionException **extends** java.lang.Exception

Thrown to alert application that option parsing failed and the user should be notified. The default constructor handles this exception automatically so that most applications can ignore it.

CONSTRUCTORS
public `Options.OptionException`( )

Package ws

The `ws` package is a plugin for an IBM Research Web Intermediaries (WBI) proxy to implement the client side of Snowflake/SDSI-based web authorization. (See `http://www.almaden.ibm.com/cs/wbi/`.) This was a first cut, but it turns out WBI is not an easy way to write a web proxy. Jetty turned out to be much easier; that proxy appears in the `jp` package.

@author jonh@cs.dartmouth.edu

Appendix F

Experimental data

In this appendix, I present the experimental data from which the tables in Chapter 12 derive. On each page, I plot the data points observed. The plot legend groups the data points into a few categories; within each category, only the dependent variable labeled on the x axis changes from one experiment to the next. The *variables* section describes the parameters that characterize each category, and the *constants* section lists the parameters that were held constant for every data point plotted on that page.

Each plot is accompanied by a small plot of the coefficients of variation (C.V.s). Each point on the C.V. plot is the C.V. of the nine runs with identical values indicated by the symbol and x-value of the plotted point. The C.V.s are presented to give an idea about the noise present in an individual experimental configuration.

I explored different dimensions of the parameter space to extract meaningful measurements of the system. The index in Table F.1 connects the summary table in Chapter 12 to the experiment number of the raw data shown on the following pages. For example, Table 12.6 measures the costs associated with authorizing a Snowflake HTTP client to a server; Table F.1 indicates that experiments 1 and 2 explore the corresponding part of the experimental parameter space. By examining the variables in experiments 1 and 2, we see that the per-byte copy cost is determined by varying the fileLength and computing the slope (b_1). The costs associated with "signed," "identical," and "MAC opt" authorization are discovered by varying the identicalRequests and useMacs experimental variables. The performance difference due to network locality is measured by performing the same set of experiments on a local machine versus a remote machine; this difference distinguishes Experiment 1 from Experiment 2.

I define each experimental variable below.

authenticateServer (boolean) In Snowflake experiments, the client expects and verifies the server's proof of the document's authenticity, and maps the issuer of that proof to a final principal in the client's `Prover`. A *final principal* is one that the client controls, such as a public key for which the `Prover` has the corresponding private key.

cacheContext (boolean) In SSL experiments, the SSL context object is reused for each request. This reuse amortizes the cost of loading certificates.

cacheSessions (boolean) In SSL experiments, the SSL session-caching optimization is

Table	Experiments
Table 12.2	23, 24
Table 12.3	25, 26
Table 12.4	17, 18, 3, 4
Table 12.5	10, 11
Table 12.6	1, 2
Table 12.7	19, 20
Table 12.8	13, 14, 15, 16
Table 12.9	8, 9

Table F.1: *The correspondence between summary tables and the experimental data they summarize*

enabled, amortizing the public-key encryptions required to establish a session.

cacheSigns (boolean) In Snowflake experiments, the server caches its signatures on documents until the modification date on the document file changes.

client When set to "fastget," the client is my trivial C HTTP client. Otherwise, the client is my Java client using Jetty stream- and header-parsing tools.

clientCachesProofs (boolean) In Snowflake RMI experiments, the client caches any proofs it generates in its `Prover`, amortizing the cost of the public-key encryption used to sign the delegation.

experimentType When set to "RMIExp," the experimental operation is an RMI transaction; otherwise, it is an HTTP transaction.

fileLength (integer, bytes) The length of user data returned by the experimental operation.

identicalRequests (boolean) In HTTP experiments, when this variable is false, the client increments the value in an extraneous header to ensure that the request differs from previous requests so that a proof of the authority of the previous request cannot be directly reused for the current request.

locality When this value is "local," the client and server process are colocated on the same host; when it is "remote," they are separated by a shared 10Mbps Ethernet segment.

numberOfConnections (integer) This value indicates the number of times the client connects to the server in a single run. A "run" is the set of experimental operations whose total wall-clock runtime appears as a data point in the plots that follow.

port (integer) The port number of the experimental server handling the request; this is an internal value determined by the socket and server variables.

protocol In HTTP experiments, this value indicates whether I use HTTP/1.0 or HTTP/1.1 requests.

registryService When this value is "TestRMIServer0," the RMI calls are transmitted on plain TCP sockets. When it is "TestRMIServer2," the RMI calls are transmitted using my `SshSocketFactory` sockets. When it is "TestRMIServer3," the calls use `SshSocketFactory` sockets, and the authority of the client is challenged using the Snowflake protocol.

requestsPerConnection (integer) In an HTTP experiment, the number of requests sent over a single connection before the connection is discarded. This variable is always 1 if the protocol is not HTTP/1.1.

server In an HTTP experiment, a value of "apache" indicates that the HTTP server is Apache. A value of "simple" indicates a simple Java web server using the `java.io.Socket` interface and trivial request parsing. A value of "Jetty" indicates a Java Jetty server with either the standard Jetty file handler or our file servlet adapted to understand Snowflake HTTP.

serverCachesProofs (boolean) In an RMI experiment, the server caches authorization proofs received from the client to amortize transmission, parsing, and verification time.

signFiles (boolean) In Snowflake experiments, the server signs delegations proving the authority of files it serves to the client.

socket In HTTP experiments, a value of "plain" indicates a plain TCP socket, and a value of "SSL" indicates an SSL socket.

uri (string) A value derived from fileLength used internally to execute HTTP experiments; it is irrelevant for purposes of analysis.

useMacs (boolean) In Snowflake HTTP experiments, the client requests a secret MAC to enable inexpensive hash-based request authorization.

useSnowflake (boolean) In HTTP experiments, indicates that Snowflake is used to authorize the client's request.

Experiment 1

Variables:

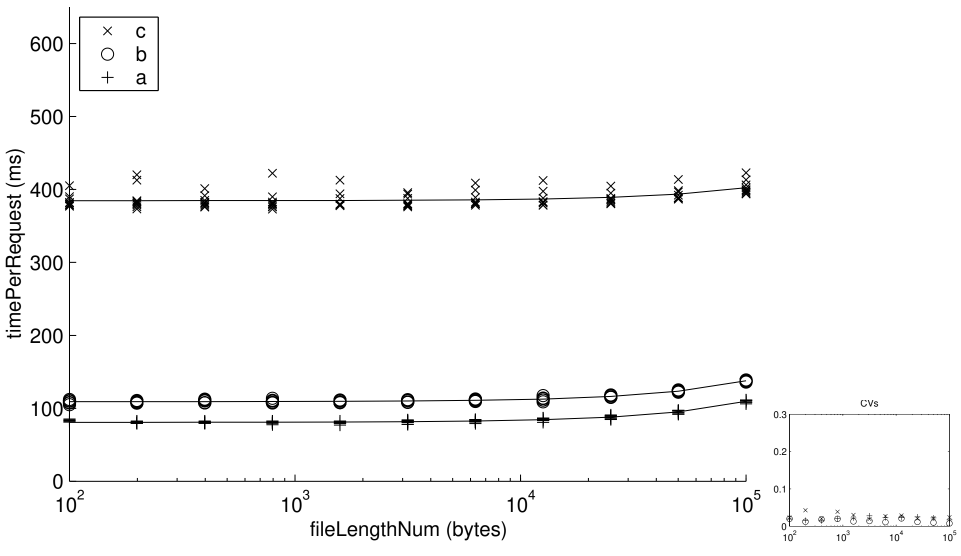
c identicalRequests=false useMacs=false authenticateServer=false
σ=10.4ms R^2=0.21 b_0=384.61 $\pm$ 0.0 ms b_1=186.98 $\pm$ 0.0 ms/MB

b identicalRequests=false useMacs=true authenticateServer=false
σ= 1.6ms R^2=0.97 b_0=109.15 $\pm$ 0.0 ms b_1=299.40 $\pm$ 0.0 ms/MB

a identicalRequests=true useMacs=true authenticateServer=false
σ= 1.9ms R^2=0.95 b_0=80.79 $\pm$ 0.0 ms b_1=301.49 $\pm$ 0.0 ms/MB

Constants:

authenticateServer	false
cacheSigns	false
numberOfConnections	10
port	8041
protocol	1.0
requestsPerConnection	1
server	simple
signFiles	false
socket	plain
useSnowflake	true
network locality	local



Experiment 2

Variables:

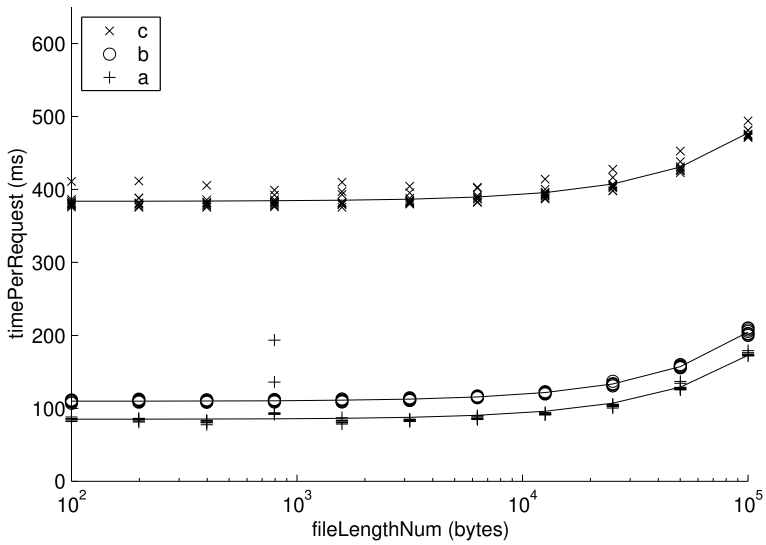
c identicalRequests=false useMacs=false authenticateServer=false
σ= 8.8ms R^2=0.91 b_0=383.77 ± 0.0 ms b_1=982.02 ± 0.0 ms/MB

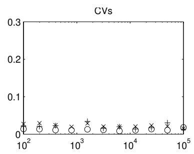
b identicalRequests=false useMacs=true authenticateServer=false
σ= 1.7ms R^2=1.00 b_0=109.67 ± 0.0 ms b_1=994.38 ± 0.0 ms/MB

a identicalRequests=true useMacs=true authenticateServer=false
σ=12.7ms R^2=0.81 b_0=85.04 ± 0.0 ms b_1=918.10 ± 0.0 ms/MB

Constants:

authenticateServer	false
cacheSigns	false
numberOfConnections	10
port	8041
protocol	1.0
requestsPerConnection	1
server	simple
signFiles	false
socket	plain
useSnowflake	true
network locality	remote





Experiment 3

Variables:

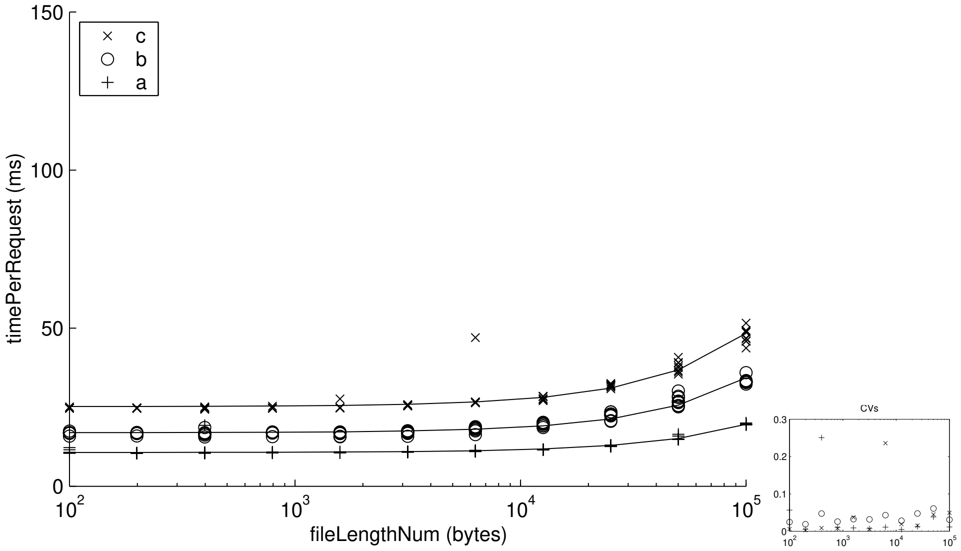
c server=Jetty
σ= 2.3ms R^2=0.90 b_0=25.22 $\pm$ 0.0 ms b_1=242.83 $\pm$ 0.0 ms/MB

b server=simple
σ= 1.0ms R^2=0.96 b_0=16.95 $\pm$ 0.0 ms b_1=181.94 $\pm$ 0.0 ms/MB

a server=apache
σ= 0.9ms R^2=0.89 b_0=10.70 $\pm$ 0.0 ms b_1=93.16 $\pm$ 0.0 ms/MB

Constants:

cacheContext	true
cacheSessions	true
numberOfConnections	20
protocol	1.0
requestsPerConnection	1
socket	plain
useSnowflake	false
network locality	local



Experiment 4

Variables:

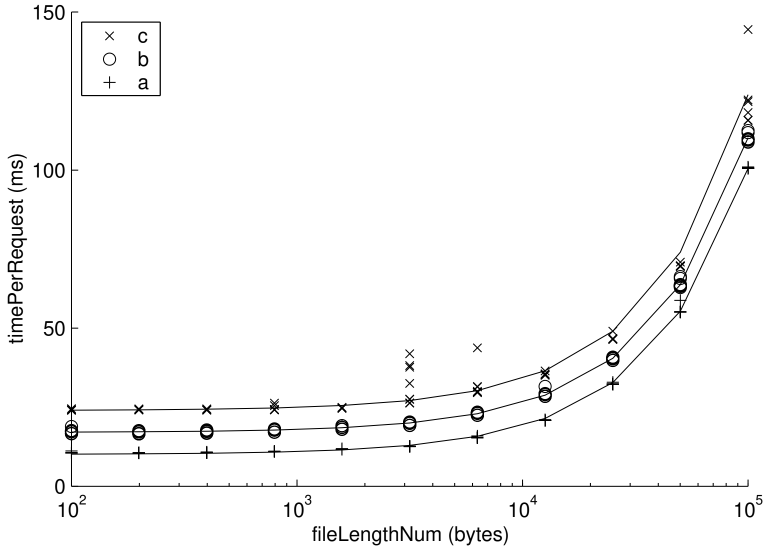
c server=Jetty
σ= 6.1ms R^2=0.96 b_0=23.98 ± 0.0 ms b_1=1047.05 ± 0.0 ms/MB

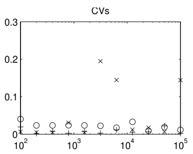
b server=simple
σ= 0.7ms R^2=1.00 b_0=17.07 ± 0.0 ms b_1=976.40 ± 0.0 ms/MB

a server=apache
σ= 0.5ms R^2=1.00 b_0=10.09 ± 0.0 ms b_1=948.42 ± 0.0 ms/MB

Constants:

cacheContext	true
cacheSessions	true
numberOfConnections	20
protocol	1.0
requestsPerConnection	1
socket	plain
useSnowflake	false
network locality	remote





Experiment 8

Variables:

b server=Jetty fileLength=000100

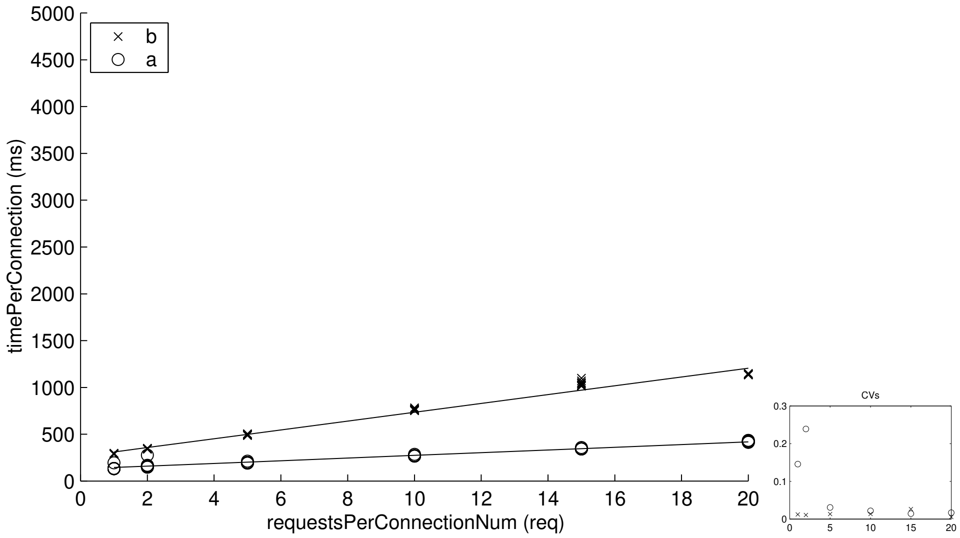
σ=45.2ms R^2=0.98 b_0=263.56 ± 0.2 ms b_1=47.13 ± 0.0 ms/req

a server=apache fileLength=000100

σ=19.4ms R^2=0.96 b_0=130.41 ± 0.0 ms b_1=14.44 ± 0.0 ms/req

Constants:

cacheContext	true
cacheSessions	true
fileLength	000100
numberOfConnections	10
protocol	1.1
socket	SSL
uri	/timing/data-000100.txt
useSnowflake	false
network locality	local



Experiment 9

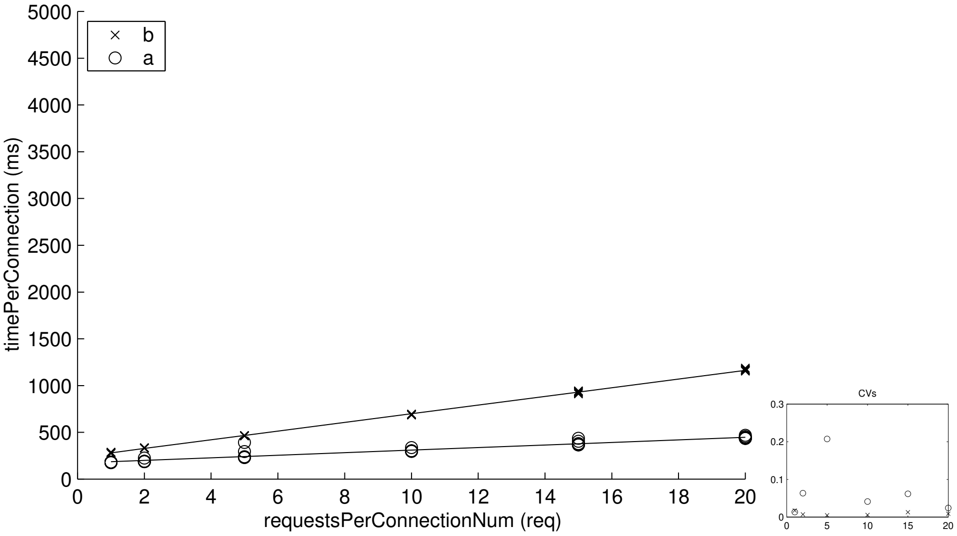
Variables:

b server=Jetty fileLength=000100
σ= 8.5ms R^2=1.00 b_0=235.04 $\pm$ 0.0 ms b_1=46.39 $\pm$ 0.0 ms/req

a server=apache fileLength=000100
σ=26.0ms R^2=0.93 b_0=172.62 $\pm$ 0.1 ms b_1=13.73 $\pm$ 0.0 ms/req

Constants:

cacheContext	true
cacheSessions	true
fileLength	000100
numberOfConnections	10
protocol	1.1
socket	SSL
uri	/timing/data-000100.txt
useSnowflake	false
network locality	remote



Experiment 10

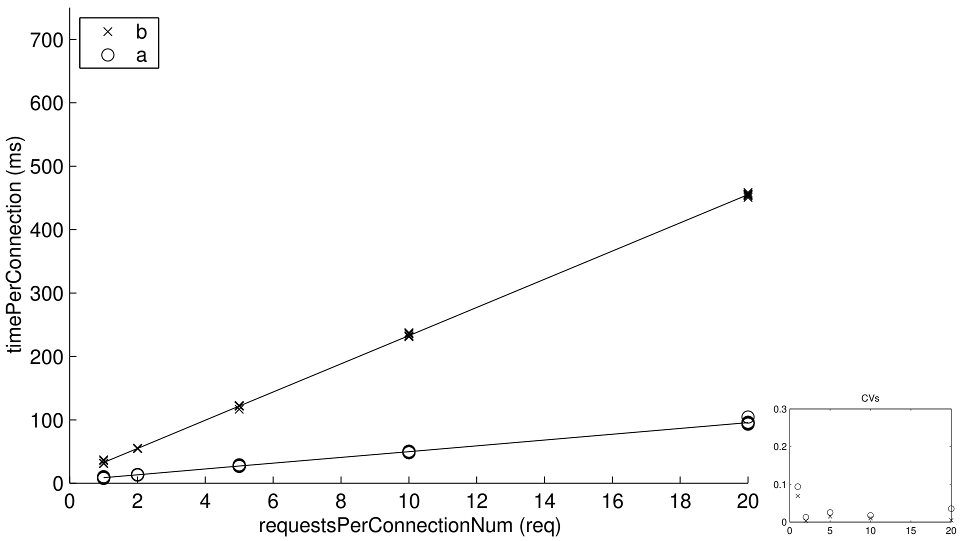
Variables:

b server=Jetty fileLength=000100
σ= 2.0ms R^2=1.00 b_0=10.39 ± 0.0 ms b_1=22.24 ± 0.0 ms/req

a server=apache fileLength=000100
σ= 1.6ms R^2=1.00 b_0= 4.27 ± 0.0 ms b_1= 4.56 ± 0.0 ms/req

Constants:

cacheContext	true
cacheSessions	true
numberOfConnections	10
protocol	1.1
socket	plain
useSnowflake	false
network locality	local



Experiment 11

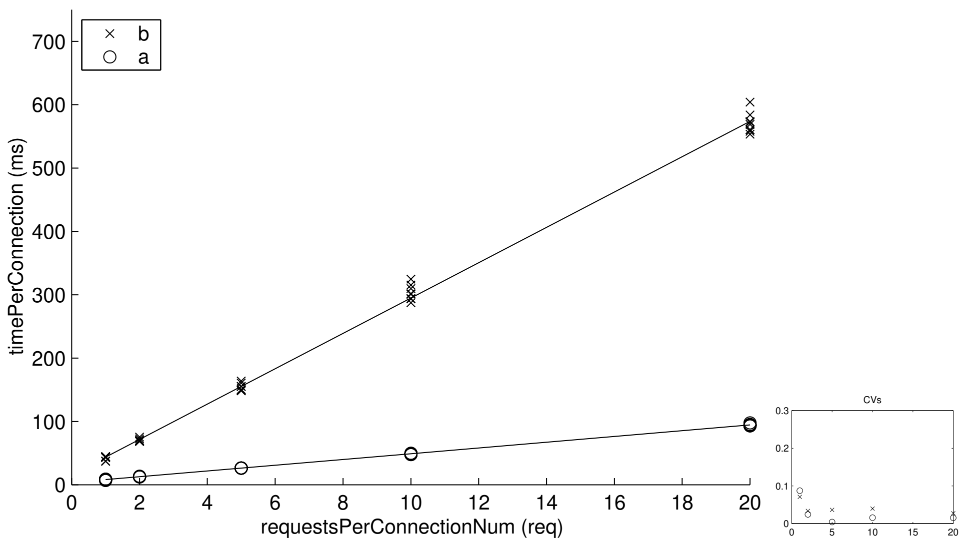
Variables:

b server=Jetty fileLength=000100
σ= 9.9ms R^2=1.00 b_0=15.76 ± 0.0 ms b_1=27.90 ± 0.0 ms/req

a server=apache fileLength=000100
σ= 0.8ms R^2=1.00 b_0= 3.64 ± 0.0 ms b_1= 4.54 ± 0.0 ms/req

Constants:

cacheContext	true
cacheSessions	true
numberOfConnections	10
protocol	1.1
socket	plain
useSnowflake	false
network locality	remote



Experiment 13

Variables:

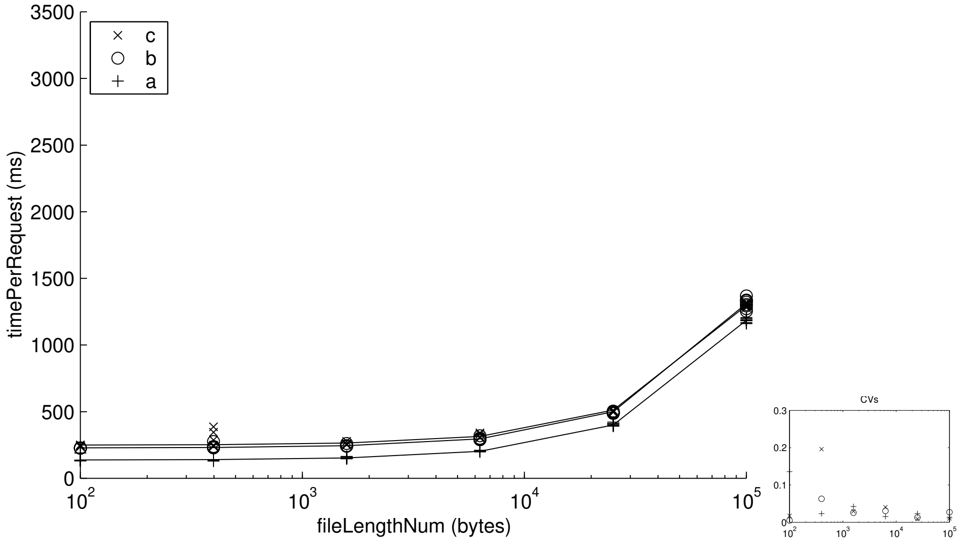
c server=apache cacheContext=false cacheSessions=false
σ=25.5ms R^2=1.00 b_0=248.43 $\pm$ 0.0 ms b_1=10990.02 $\pm$ 0.0 ms/MB

b server=apache cacheContext=true cacheSessions=false
σ=16.4ms R^2=1.00 b_0=227.30 $\pm$ 0.0 ms b_1=11367.80 $\pm$ 0.0 ms/MB

a server=apache cacheContext=true cacheSessions=true
σ=10.9ms R^2=1.00 b_0=136.68 $\pm$ 0.0 ms b_1=11002.26 $\pm$ 0.0 ms/MB

Constants:

numberOfConnections	10
protocol	1.1
requestsPerConnection	1
socket	SSL
useSnowflake	false
network locality	local



Experiment 14

Variables:

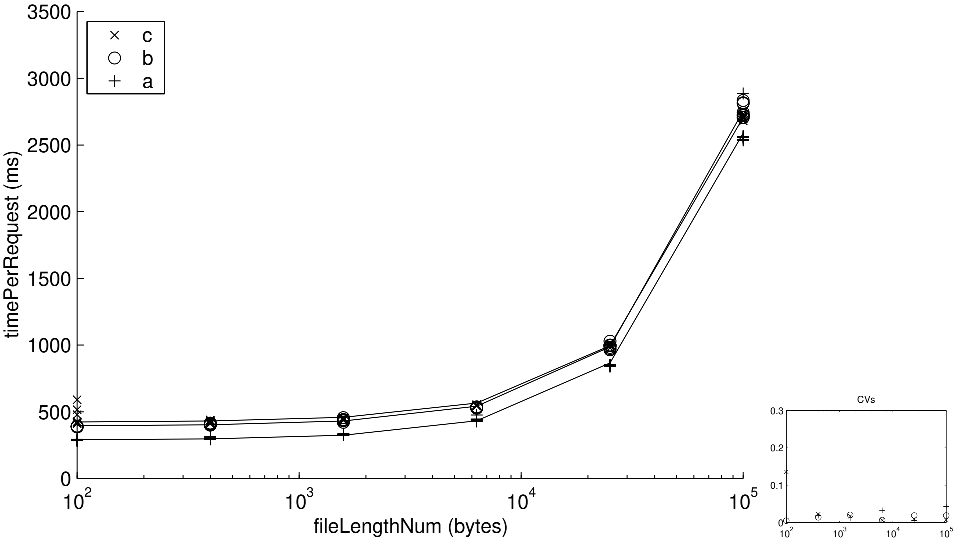
c server=Jetty cacheContext=false cacheSessions=false
σ=30.6ms R^2=1.00 b_0=422.05 ± 0.0 ms b_1=23943.09 ± 0.0 ms/MB

b server=Jetty cacheContext=true cacheSessions=false
σ=23.0ms R^2=1.00 b_0=393.45 ± 0.0 ms b_1=24760.23 ± 0.0 ms/MB

a server=Jetty cacheContext=true cacheSessions=true
σ=45.0ms R^2=1.00 b_0=288.01 ± 0.1 ms b_1=24117.47 ± 0.0 ms/MB

Constants:

numberOfConnections	10
protocol	1.1
requestsPerConnection	1
socket	SSL
useSnowflake	false
network locality	local



Experiment 15

Variables:

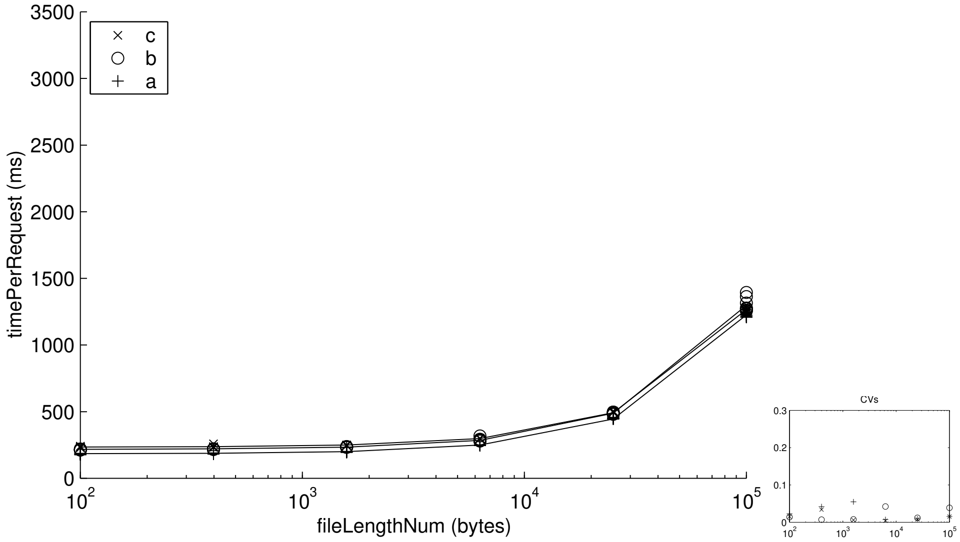
c server=apache cacheContext=false cacheSessions=false
σ= 9.0ms R^2=1.00 b_0=233.56 ± 0.0 ms b_1=10815.11 ± 0.0 ms/MB

b server=apache cacheContext=true cacheSessions=false
σ=20.6ms R^2=1.00 b_0=216.44 ± 0.0 ms b_1=11331.84 ± 0.0 ms/MB

a server=apache cacheContext=true cacheSessions=true
σ= 9.8ms R^2=1.00 b_0=184.08 ± 0.0 ms b_1=10916.32 ± 0.0 ms/MB

Constants:

numberOfConnections	10
protocol	1.1
requestsPerConnection	1
socket	SSL
useSnowflake	false
network locality	remote



Experiment 16

Variables:

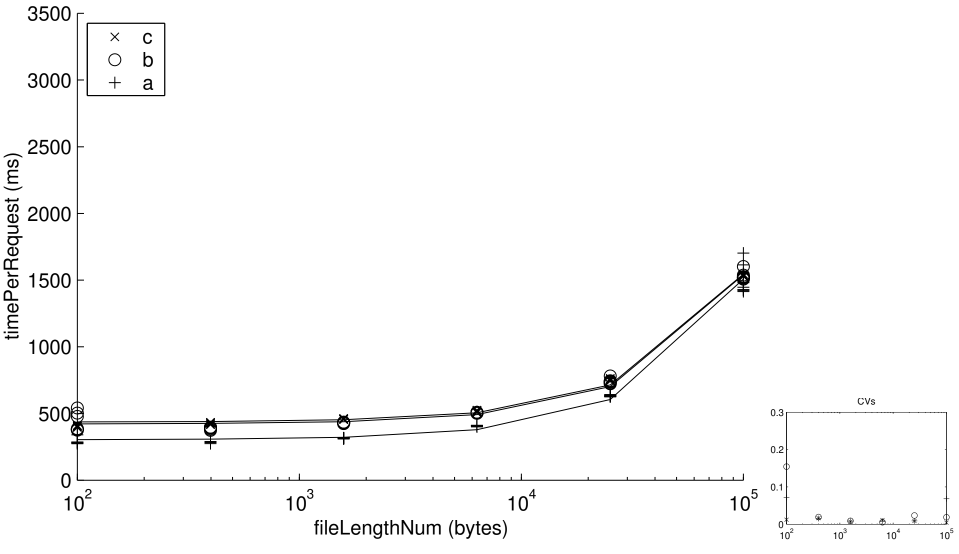
c server=Jetty cacheContext=false cacheSessions=false
σ=23.1ms R^2=1.00 b_0=436.01 $\pm$ 0.0 ms b_1=11624.15 $\pm$ 0.0 ms/MB

b server=Jetty cacheContext=true cacheSessions=false
σ=37.0ms R^2=0.99 b_0=421.37 $\pm$ 0.1 ms b_1=11723.18 $\pm$ 0.0 ms/MB

a server=Jetty cacheContext=true cacheSessions=true
σ=45.9ms R^2=0.99 b_0=303.33 $\pm$ 0.1 ms b_1=12624.89 $\pm$ 0.0 ms/MB

Constants:

numberOfConnections	10
protocol	1.1
requestsPerConnection	1
socket	SSL
useSnowflake	false
network locality	remote



Experiment 17

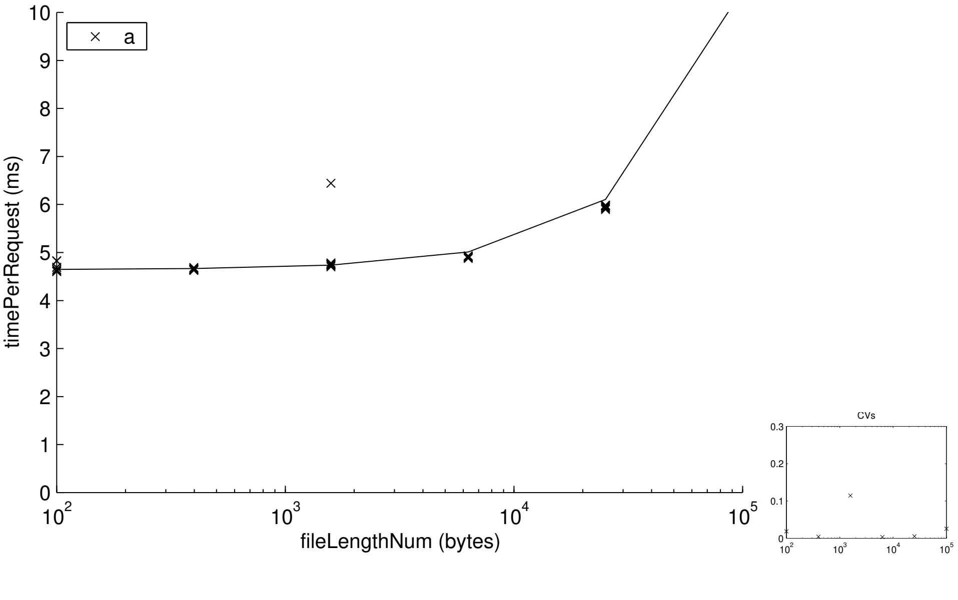
Variables:

a server=apache

σ= 0.3ms R^2=0.98 b_0= 4.64 ± 0.0 ms b_1=61.05 ± 0.0 ms/MB

Constants:

client	fastget
numberOfConnections	200
protocol	1.1
server	apache
network locality	local



Experiment 18

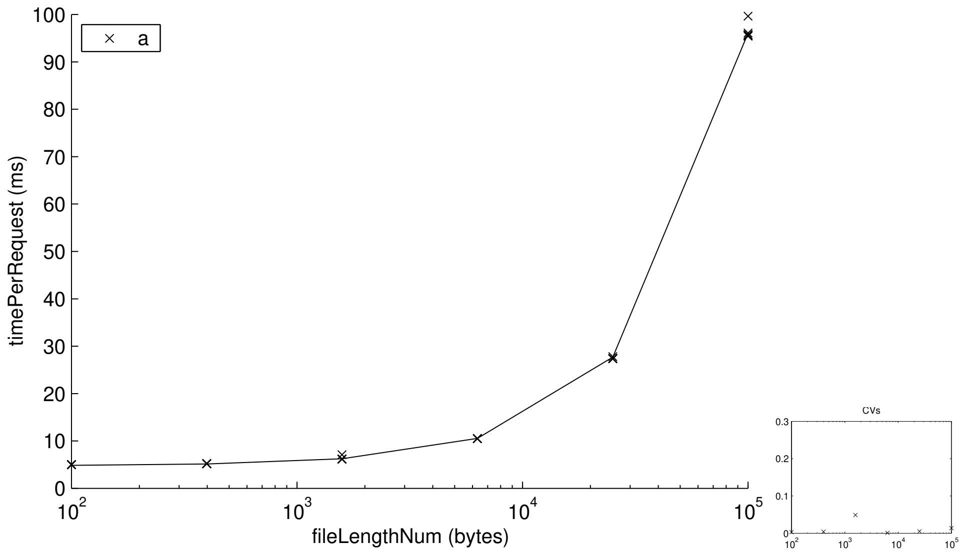
Variables:

a server=apache

σ= 0.6ms R^2=1.00 b_0= 4.79 ± 0.0 ms b_1=956.89 ± 0.0 ms/MB

Constants:

client	fastget
numberOfConnections	200
protocol	1.1
server	apache
network locality	remote



Experiment 19

Variables:

d authenticateServer=true signFiles=true cacheSigns=false
σ= 9.6ms R^2=0.95 b_0=485.52 $\pm$ 0.0 ms b_1=1197.81 $\pm$ 0.0 ms/MB

b authenticateServer=false signFiles=true cacheSigns=false
σ= 7.0ms R^2=0.93 b_0=425.38 $\pm$ 0.0 ms b_1=764.17 $\pm$ 0.0 ms/MB

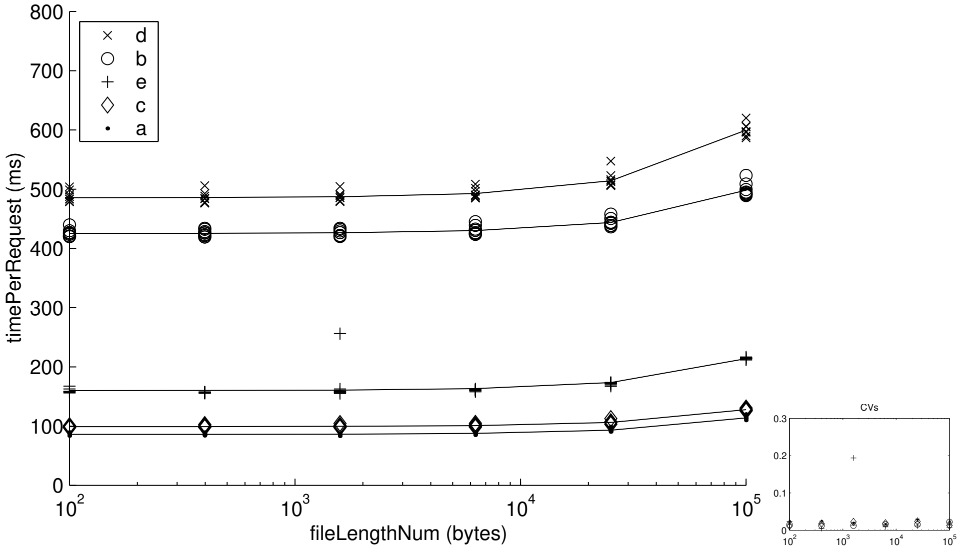
e authenticateServer=true signFiles=true cacheSigns=true
σ=13.6ms R^2=0.68 b_0=159.99 $\pm$ 0.0 ms b_1=563.97 $\pm$ 0.0 ms/MB

c authenticateServer=false signFiles=true cacheSigns=true
σ= 1.9ms R^2=0.97 b_0=99.13 $\pm$ 0.0 ms b_1=299.95 $\pm$ 0.0 ms/MB

a authenticateServer=false signFiles=false cacheSigns=false
σ= 2.2ms R^2=0.96 b_0=86.04 $\pm$ 0.0 ms b_1=293.03 $\pm$ 0.0 ms/MB

Constants:

identicalRequests	true
numberOfConnections	10
port	8041
protocol	1.0
requestsPerConnection	1
server	simple
socket	plain
useMacs	true
useSnowflake	true
network locality	local



Experiment 20

Variables:

d authenticateServer=true signFiles=true cacheSigns=false
σ=10.7ms R^2=0.97 b_0=481.71 ± 0.0 ms b_1=1757.37 ± 0.0 ms/MB

b authenticateServer=false signFiles=true cacheSigns=false
σ= 9.0ms R^2=0.97 b_0=426.82 ± 0.0 ms b_1=1397.21 ± 0.0 ms/MB

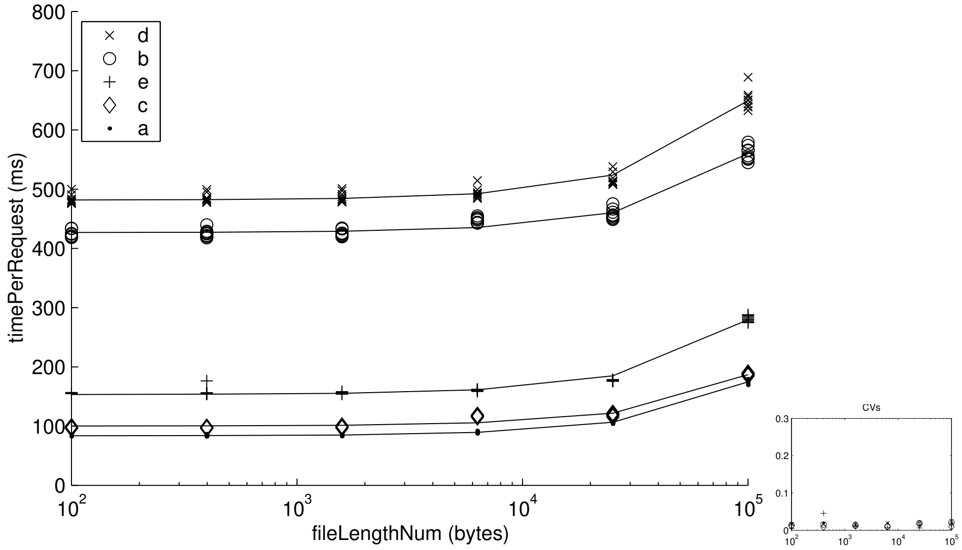
e authenticateServer=true signFiles=true cacheSigns=true
σ= 5.2ms R^2=0.99 b_0=153.42 ± 0.0 ms b_1=1321.35 ± 0.0 ms/MB

c authenticateServer=false signFiles=true cacheSigns=true
σ= 5.5ms R^2=0.97 b_0=100.11 ± 0.0 ms b_1=910.74 ± 0.0 ms/MB

a authenticateServer=false signFiles=false cacheSigns=false
σ= 1.9ms R^2=1.00 b_0=83.64 ± 0.0 ms b_1=955.37 ± 0.0 ms/MB

Constants:

identicalRequests	true
numberOfConnections	10
port	8041
protocol	1.0
requestsPerConnection	1
server	simple
socket	plain
useMacs	true
useSnowflake	true
network locality	remote



Experiment 23

Variables:

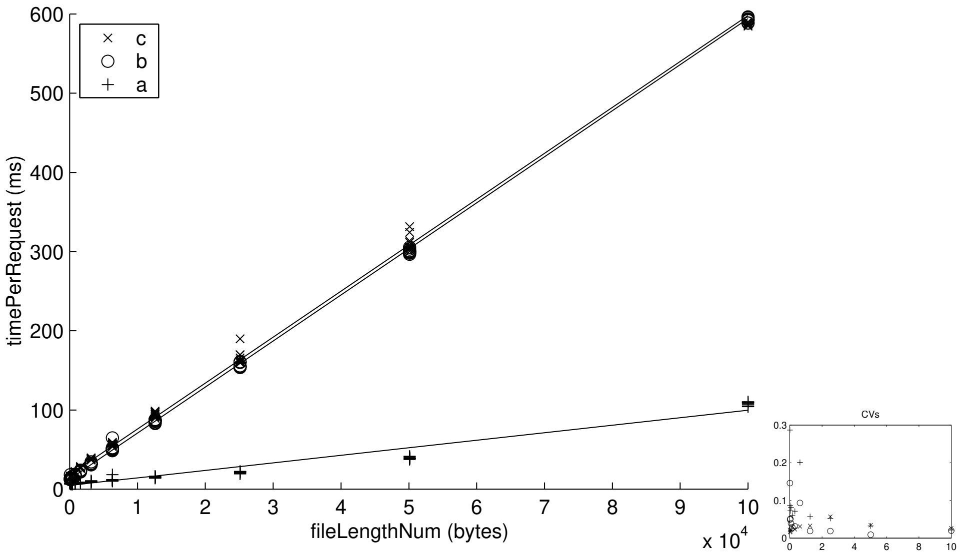
c registryService=TestRMIServer3
σ= 6.7ms R^2=1.00 b_0=17.90 $\pm$ 0.0 ms b_1=6083.68 $\pm$ 0.0 ms/MB

b registryService=TestRMIServer2
σ= 4.4ms R^2=1.00 b_0=12.67 $\pm$ 0.0 ms b_1=6097.16 $\pm$ 0.0 ms/MB

a registryService=TestRMIServer0
σ= 5.7ms R^2=0.96 b_0= 4.78 $\pm$ 0.0 ms b_1=994.51 $\pm$ 0.0 ms/MB

Constants:

experimentType	RMIExp
numberOfConnections	100
port	8143
requestsPerConnection	1
network locality	local



Experiment 24

Variables:

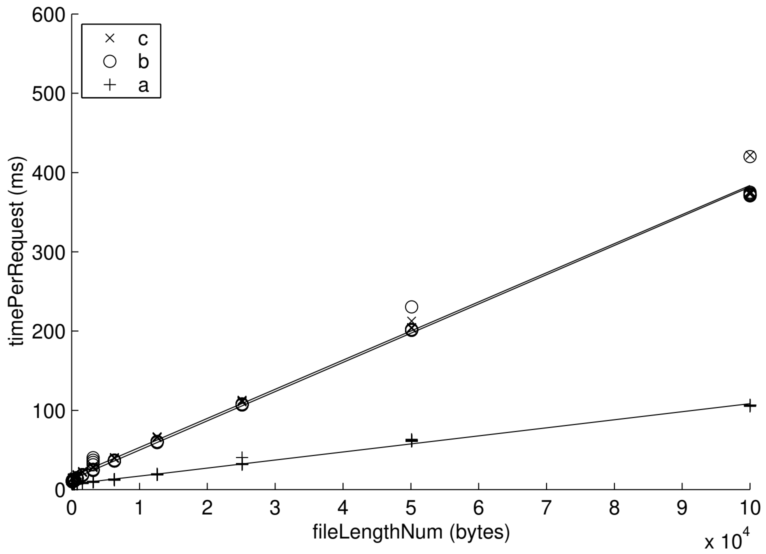
c registryService=TestRMIServer3
σ= 5.3ms R^2=1.00 b_0=16.44 ± 0.0 ms b_1=3852.20 ± 0.0 ms/MB

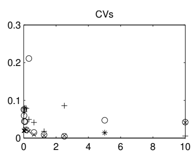
b registryService=TestRMIServer2
σ= 6.6ms R^2=1.00 b_0=12.98 ± 0.0 ms b_1=3867.08 ± 0.0 ms/MB

a registryService=TestRMIServer0
σ= 1.7ms R^2=1.00 b_0= 6.94 ± 0.0 ms b_1=1063.75 ± 0.0 ms/MB

Constants:

experimentType	RMIExp
numberOfConnections	100
port	8143
requestsPerConnection	1
network locality	remote





Experiment 25

Variables:

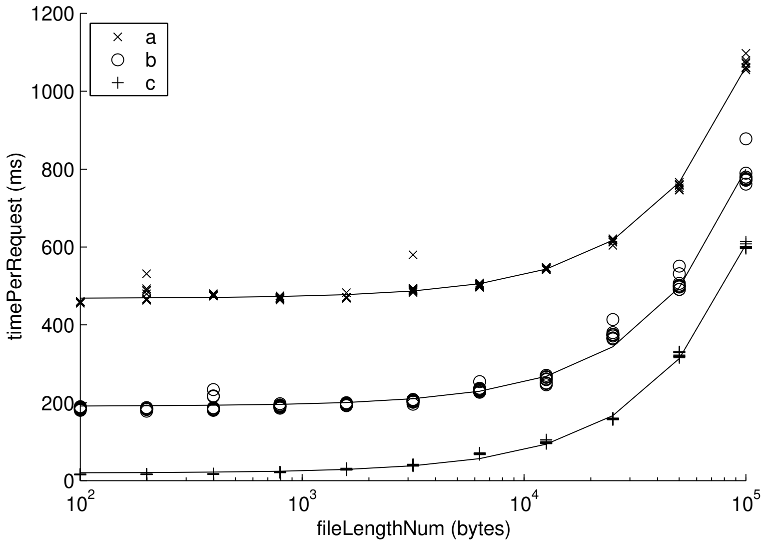
a clientCachesProofs=false serverCachesProofs=false
σ=14.4ms R^2=0.99 b_0=468.29 ± 0.0 ms b_1=6235.17 ± 0.0 ms/MB

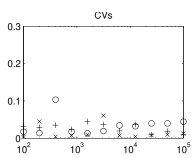
b clientCachesProofs=true serverCachesProofs=false
σ=18.8ms R^2=0.99 b_0=191.27 ± 0.0 ms b_1=6377.39 ± 0.0 ms/MB

c clientCachesProofs=true serverCachesProofs=true
σ= 7.1ms R^2=1.00 b_0=19.77 ± 0.0 ms b_1=6149.06 ± 0.0 ms/MB

Constants:

experimentType	RMIExp
numberOfConnections	30
port	8143
registryService	TestRMIServer3
requestsPerConnection	1
network locality	local





Experiment 26

Variables:

a clientCachesProofs=false serverCachesProofs=false
σ= 5.1ms R^2=1.00 b_0=411.43 ± 0.0 ms b_1=3798.24 ± 0.0 ms/MB

b clientCachesProofs=true serverCachesProofs=false
σ= 5.5ms R^2=1.00 b_0=140.97 ± 0.0 ms b_1=3796.35 ± 0.0 ms/MB

c clientCachesProofs=true serverCachesProofs=true
σ= 4.4ms R^2=1.00 b_0=16.94 ± 0.0 ms b_1=3790.53 ± 0.0 ms/MB

Constants:

experimentType	RMIExp
numberOfConnections	15
port	8143
registryService	TestRMIServer3
requestsPerConnection	1
network locality	remote