InheritedProperties

Property Inheritance

• Property Inheritance
  • Background
    • What's This?
    • What Is Property Inheritance?
    • What's Driving This?
    • What This Design Is and What It Isn't
    • How Is This Design Different?
    • Inheritable Properties Today
  • Design
    • Differentiating 'Inheritable' Vs. 'Normal' Properties
    • General Inheritance Rules
    • Repository Inheritance Rules
    • Working Copy Inheritance Rules
    • Inherited Properties Cache
    • Externals
    • Authentication
    • Merging Inherited Properties
    • Subcommand Changes
    • API Changes
    • New APIs and Public Structs
    • New Capabilities
    • Drawbacks

Background

What's This?

Some ideas on how we might make versioned properties inheritable.

What Is Property Inheritance?

Property inheritance provides a mechanism to find the versioned properties set on the path-wise parents of a given path. Conversely it can be viewed as a mechanism by which a versioned property set on a given path also applies to that path's children.

What's Driving This?

Desire for some form of inherited properties has existed almost from the dawn of Subversion. This is in response to a recent proposal regarding Server Dictated Configuration (see http://svn.haxx.se/dev/archive-2012-01/0032.shtml). That proposal made use of a new mechanism by which server dictated configurations (notably auto-props and global-ignores) would be communicated from the server to the client. In the feedback on the proposal several people pointed out that versioned properties provide a possible alternative solution (and that the TortoiseSVN project was already using pseudo-inheritable properties to address some of the same problems – see http://tortoisesvn.net/docs/nightly/TortoiseSVN_en/tsvn-dug-propertypage.html). Despite its origins in the server defined configuration work, this wiki describes only a generic inheritable properties design (though my the ultimate goal is to use these inheritable properties to implement the server dictated configuration feature – pburba).

What This Design Is and What It Isn't

This design provides the bare minimum to support the basics of inherited properties:

• A way to get a path's inherited properties from the repository.
• A way to cache a working copy's inherited properties locally so that disconnected WC operations remain disconnected.

That's it, it's really just a small extension of our existing versioned property capabilities. Since any versioned property can be inherited, there is no concept of setting "inheritable" properties. Our existing APIs are sufficient. If your own personal vision of inherited properties includes something outside of these two bullets (e.g. merging inherited properties), then that is outside of this design's scope (at least initially).

How Is This Design Different?

It's not really. A lot of the ideas here come from the dev list, #svn-dev, and conversations with other devs. The difference, if there is any, is that this design aims to be as simple as possible and steer clear of complications which have thwarted inheritable properties in the past. If there is more than one reasonable behavior to choose from, it usually goes with the one that is simpler to explain/understand/implement/maintain. This means that not everyone will be happy, it may not cover every use-case, but I hope it covers most of them – pburba.

Inheritable Properties Today

Subversion has supported merge-tracking using the svn:mergeinfo property since the 1.5 release. The svn:mergeinfo property is inheritable in some merge-sensitive contexts, notably 'svn merge', 'svn log -g', and 'svn mergeinfo'. For example, say we have this simple branch structure at r700 in our repository:
And suppose the mergeinfo ‘projX/trunk:509-612’ is set on ‘projX/branches’. If we perform a merge targeting ‘projX/branches/src/foo.c’, the merge logic considers the svn:mergeinfo property on this target path to effectively be ‘projX/trunk/src/foo.c:509-612’. However, unlike a true inheritable property however, svn:mergeinfo is not inheritable outside of merge-tracking aware contexts. For example, the propget and proplist subcommands recognize the svn:mergeinfo when it is explicitly set on a path, but does not “see” it on those paths’ descendants:

```bash
>svn pg svn:mergeinfo -v projX/branches Properties on 'projX/branches':
  svn:mergeinfo /projX/trunk:509-612
>svn pg svn:mergeinfo -v branches/projX/src/foo.c
```

**Design**

**Differentiating 'Inheritable' Vs. 'Normal' Properties**

This is easy, there is no difference, there is no such thing as an "inheritable" property. Versioned properties, both Subversion's reserved properties and custom user properties, can be interpreted as inheritable, but otherwise function as they always have in terms of repository storage, setting properties, valid property names and values, etc.. What is proposed here is a new way of looking at the familiar versioned property. The only differentiation that is important as far as the design is this: Is a property value **inherited or explicit**? A path may have property 'PropName' explicitly set on it or property values may be inherited by a child path from some parent path(s) which also have 'PropName' explicitly set on them.

**General Inheritance Rules**

A path (we'll refer to this as the 'child' path from here on) inherits a given property from any of the child's path-wise ancestors (the 'parent' paths) on which the given property is explicitly set. This means that a given path can inherit multiple values for a given property from different parents. Further, the child path may also have the given property explicitly set on itself. This obviously differs from svn:mergeinfo, where inheritance only occurs when a child lacks explicit mergeinfo and only the nearest parent's mergeinfo is inherited. The new APIs will provide a list of parent paths/URLs and the properties inherited from each. It will be up to the caller, user, script, etc.., to decide what it wants to do with this information. The goal here is maximum flexibility. It will be possible to implement a mergeinfo-like inheritance scheme or to merge multiple values together, again, whatever the consumer wants to do.

If a child path which inherits a property is copied, the inherited properties are not copied with it. The new destination path inherits properties from its new parents. This means that depending on the copy destination the new path may or may not inherit the same property, and even if it does, it may inherit a different value. This puts the onus on users to set their inheritable properties as close to the root of the repository tree as makes sense for the property in question (while considering any authorization restrictions that might thwart inheritance). Have a property you want to apply to the whole repository? Set it on the root. Want it to apply only to a single project? Set it on the root of the project. Note that if a path has an inheritable property explicitly set on it, the property is copied just like any other versioned property.

[1] Generic inherited properties also differ from mergeinfo in that the property value a child inherits from a parent is not based on the path-difference between the parent and child. The exact property value on the parent is the value the child will inherit.

**Repository Inheritance Rules**

Inheritance of properties within the repository is pretty straightforward, for a given property 'PropName':

- A repository child_path@REV may inherit one or more values of the 'PropName' property from any of the child's parents (i.e. its path-wise ancestors @REV) which have the 'PropName' property explicitly set on them.

**Working Copy Inheritance Rules**

- Inheritance within the working copy is a bit more complicated. For a given property 'PropName' there are three basic cases of WC inheritance:
  1. If the peg and operative revisions are for the path in question are anything other than unspecified, then the repository is contacted to obtain the path's inherited properties. This is analogous to the repository inheritance model.
  2. If the peg and operative revisions are unspecified, then the working copy child path inherits properties from its parents in the **actual**
tree, up to nearest working copy root (this is termed "default working copy inheritance"). Once the working copy root is reached, a WC child path may also inherit properties from repository parents via the inherited properties cache, which caches the properties inherited by the base node at the WC root (see the Inherited Properties Cache section). "Working copy roots" here are defined as:

- A directory whose parent directory is not a working copy.
- A directory or file which is switched relative to its immediate parent.
- The root of a directory external.
- A file external.

These default WC inheritance rules have some important implications:

3. A path, in an unmodified working copy at a uniform revision N, inherits the same properties that the base of that path would inherit in the repository at revision N.

4. A path in a modified working copy effectively inherits properties from the proto-revision that any modified WC represents (a set of changes that could potentially be committed as a new revision).

5. Mixed-revision working copies are similar to #2 except that there is no proto-revision. So unlike svn:mergeinfo and like tsvn:auto-props, inheritance across mixed-revision boundaries in the working copy is allowed by default.

Leaving Julian's questions for now...

### Inherited Properties Cache

A child path that inherits a property from its parent may not have ready access to that parent in the working copy (e.g. the root of the working copy is a subtree of the parent path). To ensure that traditionally disconnected operations (i.e. those that require no access to the repository, like 'svn add') remain disconnected, we will maintain a cache of properties inherited by the root of the working copy. Whenever a new working copy is checked out, any properties inherited by the root of the working copy will be cached in the working copy. If a subtree within a working copy is switched, a separate cache will be created for the root of that subtree. File externals will have their own cache as well. Whenever the WC is updated the cache(s) will be refreshed.

The cache will be stored in a new BLOB column in the NODES table called 'inherited_props'. If the node is the base node of a WC root then this column is populated with a serialized skel of the node's inherited properties. In all other cases it is NULL.

### Externals

Directory externals are treated as self-contained working copies with their own inherited properties cache. File externals, despite not being their own WC, are also treated as WC roots and get their own cache. The end result: Property inheritance never crosses external boundaries in the WC.

### Authentication

The svn:mergeinfo property can be inherited by any child path the user has read access to, even if the user has no access to the parent path the svn:mergeinfo property is explicitly set on. Generic property inheritance is more restrictive:

- Properties can only be inherited from paths which the user has read access to.
- Inheritance from a grandparent can occur across an intermediate parent the user has no read access to, as long as the user can read the grandparent.

For example, given a repository structure like this:

```
^/Projects
^/Projects/ProjX
^/Projects/ProjX/Docs
^/Projects/ProjX/Docs/Manual
```

And this authz:

```plaintext
[/Projects]
random=r
[/Projects/ProjX]
random=
[/Projects/ProjX/Docs]
random=rw
```

If random asks via the appropriate API what properties ^/Projects/ProjX/Docs/Manual inherits, he would see the property:value pairs from ^/Projects/ProjX/Docs and ^/Projects, but not ^/Projects/ProjX.

Yes, this means that an administrator could set up a restrictive authz configuration that could thwart what might be desirable inheritance, e.g. very few users have read access to the repository root, but properties are set on the root that are intended to have repository-wide application. The assumption is that if an administrator sets inheritable properties on the repository root intending that they apply to the entire repos for all users, then he will ensure that all users have read access to the root.

### Merging Inherited Properties


This design purposefully avoids any concept of how inherited properties (possibly from multiple parents) might be merged together with explicit properties. In some cases if a path has an inheritable property set on it, then it might make sense to consider that property's value as the complete and full value for that path, end of story (e.g. svn:mergeinfo). On the other hand, it's easy to imagine use-cases where it might be useful to merge a path's explicit and inherited properties. However, both cases depend on the semantics of the particular inherited property. There is no way to develop a one-size-fits-all approach to merging inheritable properties. So while the API changes below support the ability to get a path's explicit and inherited properties, how to merge these values (if at all) will be handled on a case-by-case basis.

Subcommand Changes

Since "inherited" properties are really just a new way of looking at versioned properties, most subcommands will only notice the paths on which explicit properties are set. For example, if we have an unmodified working copy and then make a local change to a parent path's explicit property:

- `svn status` won't show any property mods on the parent's children paths.
- `svn diff` will only show the property difference on the parent path.

The are a few exceptions to this business-as-usual behavior. A few subcommands require some non-user-visible changes:

- `checkout (co)`: Populates the inherited properties cache for the WC root.
- `switch (sw)`: Creates/clears the inherited properties cache for the root of the switched/unswitched subtree.
- `update (up)`: Updates the inherited properties cache(s) for any WC roots within the operational depth of the update.
- `upgrade`: Creates an empty (but non-null) inherited properties caches for all WC roots, but doesn't populate any of them since that would require contacting the repository which upgrade shouldn't need to do. Subsequent updates will populate the caches.

Two subcommands, `propget (pget, pg)` and `proplist (plist, pl)`, will support a new option enabling users to view inherited properties on a path: `--show-inherited-props`. When used, any properties inherited by the target of the subcommand will be displayed. Properties inherited by subtrees of the target are not shown. For example: Given this repository structure with the explicit properties noted:

```
/                prop:foo=bar
branch/          prop:foo=bat
branch/src/      prop:foo=bax
branch/src/pop.c
branch/inc
branch/inc/pop.h
trunk/
trunk/src/
trunk/src/pop.c
trunk/inc
trunk/inc/pop.h
```

Without the `--show-inherited-props` option only explicit properties are shown (as has always been the case):

```
> svn pl -vR ^/branch
Properties on '%ROOT_URL%/branch':
  prop:foo
    bat
Properties on '%ROOT_URL%/branch/src':
  prop:foo
    bax
```

With the `--show-inherited-props` option, explicit properties are shown as normal, but any properties inherited by the target are also shown. For example, a recursive `propget` on `^/branch` shows the explicit properties on the target and its children, but also shows that the target inherits `prop:foo` from the root of the repository:
A target with no explicit properties might still inherit a property from a parent:

```
>svn pl -R --show-inherited-props ^/branch
Properties inherited from '%ROOT_URL%':
  prop:foo
Properties on '%ROOT_URL%/branch':
  prop:foo
Properties on '%ROOT_URL%/branch/src':
  prop:foo
```

If a path inherits a property from multiple parents, all the parents are shown:

```
>svn pg -v --show-inherited-props prop:foo ^/branch/src/pop.c
Properties inherited from '%ROOT_URL%':
  prop:foo
  bar
Properties inherited from '%ROOT_URL%/branch':
  prop:foo
  bat
Properties inherited from '%ROOT_URL%/branch/src':
  prop:foo
  bax
```

If the target is a WC path and some of the inherited properties come from the repository, those from the repository are listed by URL rather than WC path:

```
>svn co %ROOT_URL%/branch branch-WC
A branch-WC\src
A branch-WC\src\pop.c
A branch-WC\inc
A branch-WC\inc\pop.h
U branch-WC
Checked out revision 11.
>cd branch-WC
>svn pg -vR prop:foo
Properties inherited from '%ROOT_URL%':
  prop:foo
  bar
Properties on '.':
  prop:foo
  bat
Properties on 'src':
  prop:foo
  bax
```

The above examples apply only to the general case. Subversion reserved properties that we deem inheritable may exhibit differing behavior, depending on how we define inheritance merging for the property in question. For example, might show either the explicit mergeinfo on or the mergeinfo existing on nearest path-wise parent with explicit mergeinfo (or possible nothing if there was no explicit mergeinfo on the target or its parents).

We may want to support different values for --show-inherited-props, possibly:

```
--show-inherited-props=["all" (default) | "nearest"]
```

- 'all' - Shows the target's explicit properties and all path-wise parents with inheritable properties.
- 'nearest' - Shows the target's explicit properties and the nearest path-wise parent for any properties which are "not" explicitly set on the target. This would be useful for seeing inheritable properties which follow a strict override model like svn:mergeinfo.

**API Changes**
Rev svn_proplist_receiver_t to svn_proplist_receiver2_t, adding the argument apr_array_header_t *inherited_props. Also allow the possibility that the existing apr_hash_t *prop_hash argument may be null:

```c
/**
 * The callback invoked by svn_client_proplist4(). Each invocation
 * provides the regular and/or inherited properties of @a path, which is
 * either a working copy path or a URL. If @a prop_hash is not @c NULL, then
 * it maps explicit <tt>const char *</tt> property names to
 * <tt>svn_string_t *</tt> explicit property values. If @a inherited_props
 * is not @c NULL, then it is a depth-first ordered array of
 * #svn_prop_inherited_item_t * structures representing the
 * properties inherited by @a path. Use @a scratch_pool for all temporary
 * allocations.
 * @since New in 1.8.
 */
typedef svn_error_t *(*svn_proplist_receiver2_t)(
    void *baton,
    const char *path,
    apr_hash_t *prop_hash,
    apr_array_header_t *inherited_props,
    apr_pool_t *scratch_pool);
```

[JAF] What do the keys of 'inherited_prop_hash' represent if 'path' is a working copy path but some of the inherited properties are set on repository paths there above the WC root?

[PTB] JAF - It's a bit awkward, but I was thinking WC relative paths for props inherited from WC parents and URLs for props inherited from the repository (via the inherited prop cache). We obviously can't use WC keys in all cases, but using URLs in all cases won't work either because a WC parent might not exist in the repository due to local changes.

[PTB] Instead of using a hash mapping parent "paths" to a hash of properties, it probably makes more sense to use an array of structures which contain the path/URL and the property hash as members. The array would be sorted by the depth of the path/URL and would allow the caller to easily determine the nearest parent of PATH if that is all that it needs (think of a property with a svn:mergeinfo-like straight override model of inheritance where all that ever matters is the nearest parent). Done - change noted above.

Rev svn_client_proplist3 to svn_client_proplist4, adding the argument svn_boolean_t get_target_inherited_props as well as a scratch pool and use the new svn_proplist_receiver2_t callback:
* Invoke `@a receiver` with `@a receiver_baton` to return the regular explicit, and possibly the inherited, properties of `@a target`, a URL or working copy path. `@a receiver` will be called for each path encountered.

* `@a target` is a WC path or a URL.

* If `@a revision->kind` is `#svn_opt_revision_unspecified`, then get the explicit (and possibly the inherited) properties from the working copy, if `@a target` is a working copy path, or from the repository head if `@a target` is a URL. Else get the properties as of `@a revision`.

* If `@a target` is a URL, then get the properties of `@a target` from the repository head if `@a revision->kind` is `#svn_opt_revision_unspecified`, then it defaults to `#svn_opt_revision_head` for URLs or `#svn_opt_revision_working` for WC targets. Use the authentication baton cached in `@a ctx` for authentication if contacting the repository.

* If `@a depth` is `#svn_depth_empty`, list only the properties of `@a target` itself. If `@a depth` is `#svn_depth_files`, and `@a target` is a directory, list the properties of `@a target` and its file entries. If `@a depth` is `#svn_depth_infinity`, list the properties of its file entries and recurse (with `@a depth` as `#svn_depth_infinity`) on directory entries. `#svn_depth_unknown` is equivalent to `#svn_depth_empty`. All other values produce undefined results.

* `@a changelists` is an array of `const char *` changelist names, used as a restrictive filter on items whose properties are listed; that is, don't list properties on any item unless it's a member of one of those changelists. If `@a changelists` is empty (or altogether `@c NULL`), no changelist filtering occurs.

* If `@a get_target_inherited_props` is true, then also return any inherited properties when `@a receiver` is called for `@a target`. If `@a target` is a working copy path, then properties inherited by `@a target` as far as the root of the working copy are obtained from the working copy's actual property values. Properties inherited from above the working copy root come from the inherited properties cache. If `@a target` is a URL, then the inherited properties come from the repository.

* If `@a get_target_inherited_props` is false, then no inherited properties are returned to `@a receiver`.

* If `@a target` is not found, return the error `#SVN_ERR_ENTRY_NOT_FOUND`.

* `@since New in 1.8.`

```c
svn_error_t *
svn_client_proplist4(const char *target,
    const svn_opt_revision_t *peg_revision,
    const svn_opt_revision_t *revision,
    svn_depth_t depth,
    const apr_array_header_t *changelists,
    svn_boolean_t get_target_inherited_props,
    svn_proplist_receiver2_t receiver,
    void *receiver_baton,
    svn_client_ctx_t *ctx,
    apr_pool_t *result_pool,
    apr_pool_t *scratch_pool);
```
svn_client_propget5

svn_error_t *
svn_client_propget5(apr_hash_t **props,
                    apr_array_header_t **inherited_props,
                    const char *propname,
                    const char *target,  /* abspath or URL */
                    const svn_opt_revision_t *peg_revision,
                    const svn_opt_revision_t *revision,
                    svn_revnum_t *actual_revnum,
                    svn_depth_t depth,
                    const apr_array_header_t *changelists,
                    svn_client_ctx_t *ctx,
                    apr_pool_t *result_pool,
                    apr_pool_t *scratch_pool);

rev svn_ra_get_dir2 to svn_ra_get_dir3, adding the argument apr_array_header_t **inherited_props. * Also allow the possibility that the existing apr_hash_t **props argument may be null:
/**
 * If @a dirents is non @c NULL, set @a *dirents to contain all the entries
 * of directory @a path at @a revision. The keys of @a dirents will be
 * entry names (<tt>const char *</tt>), and the values dirents
 * (<tt>@c svn_dirent_t *</tt>). Use @a pool for all allocations.
 *
 * @a dirent_fields controls which portions of the <tt>@c svn_dirent_t</tt>
 * objects are filled in. To have them completely filled in just pass
 * @c SVN_DIRENT_ALL, otherwise pass the bitwise OR of all the @c SVN_DIRENT_
 * fields you would like to have returned to you.
 *
 * @a path is interpreted relative to the URL in @a session.
 *
 * If @a revision is @c SVN_INVALID_REVNUM (meaning 'head') and
 * @a *fetched_rev is not @c NULL, then this function will set
 * @a *fetched_rev to the actual revision that was retrieved. (Some
 * callers want to know, and some don't.)
 *
 * If @a props is non @c NULL, set @a *props to contain the properties of
 * the directory. This means @em all properties: not just ones controlled by
 * the user and stored in the repository fs, but non-tweakable ones
 * generated by the SCM system itself (e.g. 'wcprops', 'entryprops',
 * etc.) The keys are <tt>const char *</tt>, values are
 * (<tt>@c svn_string_t *</tt>.
 *
 * If the server advertises the #SVN_RA_CAPABILITY_INHERITED_PROPS capability
 * and if @a inherited_props is not @c NULL, then set @a *inherited_props to
 * a depth-first ordered array of #svn_prop_inherited_item_t * structures
 * representing the properties inherited by @a path at @a revision (or the
 * 'head' revision if @a revision is @c SVN_INVALID_REVNUM. If
 * @a inherited_props is not @c NULL and no inheritable properties are found,
 * then set @a *inherited_props to be an empty array. If the server does not
 * advertise the #SVN_RA_CAPABILITY_INHERITED_PROPS capability and
 * @a inherited_props is not @c NULL, then set @a *inherited_props to
 * @c NULL.
 *
 * @since New in 1.8.
 */
svn_error_t *
svn_ra_get_dir3(svn_ra_session_t *session,
               apr_hash_t **dirents,
               svn_revnum_t *fetched_rev,
               apr_hash_t **props,
               apr_array_header_t **inherited_props,
               const char *path,
               svn_revnum_t revision,
               apr_uint32_t dirent_fields,
               apr_pool_t *result_pool,
               apr_pool_t *scratch_pool);
/**
 * Fetch the contents and properties of file @a path at @a revision.
 * @a revision may be SVN_INVALID_REVNUM, indicating that the HEAD
 * revision should be used. Interpret @a path relative to the URL in
 * @a session. Use @a pool for all allocations.
 * If @a revision is @c SVN_INVALID_REVNUM and @a fetched_rev is not
 * @c NULL, then set @a *fetched_rev to the actual revision that was
 * retrieved.
 * If @a stream is non @c NULL, push the contents of the file at @a
 * stream, do not call svn_stream_close() when finished.
 * If @a props is non @c NULL, set @a *props to contain the properties of
 * the file. This means @em all properties: not just ones controlled by
 * the user and stored in the repository fs, but non-tweakable ones
 * generated by the SCM system itself (e.g. 'wcprops', 'entryprops',
 * etc.). The keys are <tt>const char *</tt>, values are
 * <tt>svn_string_t *</tt>.
 * If the server advertises the #SVN_RA_CAPABILITY_INHERITED_PROPS capability
 * and if @a inherited_props is not @c NULL, then set @a *inherited_props to
 * a depth-first ordered array of #svn_prop_inherited_item_t * structures
 * representing the properties inherited by @a path at @a revision (or the
 * 'head' revision if @a revision is @c SVN_INVALID_REVNUM). If
 * @a inherited_props is not @c NULL and no inheritable properties are found,
 * then set @a *inherited_props to an empty array. If the server does not
 * advertise the #SVN_RA_CAPABILITY_INHERITED_PROPS capability and
 * @a inherited_props is not @c NULL, then set @a *inherited_props to
 * @c NULL.
 * The stream handlers for @a stream may not perform any RA
 * operations using @a session.
 * @since New in 1.8.
 */
svn_error_t *
svn_ra_get_file2(svn_ra_session_t *session,
    const char *path,
    svn_revnum_t revision,
    svn_stream_t *stream,
    svn_revnum_t *fetched_rev,
    apr_hash_t **props,
    apr_array_header_t **inherited_props,
    apr_pool_t *result_pool,
    apr_pool_t *scratch_pool);

/* svn_fs_node_proplist to svn_fs_node_proplist2, adding the argument apr_hash_t **inherited_table_p. Also allow the possibility that the existing
apr_hash_t **table_p argument may be null:*/

/** If @a table_p is not null, then set @a *table_p to the entire property
 * list of @a path in @a root. The resulting table maps
 * property names to pointers to #svn_string_t objects containing the
 * property value.
 *
 * If @a inherited_values is not @c NULL, then set @a *inherited_values to
 * a depth-first ordered array of #svn_prop_inherited_item_t * structures
 * (the path_or_url members of which are relative filesystem paths)
 * representing the properties inherited by @a path. If @a inherited_values
 * is not @c NULL and no properties are inherited, then set
 * @a *inherited_values to an empty array.
 *
 * @since New in 1.8.
 */
svn_error_t *
svn_fs_node_proplist2(apr_hash_t **table_p,
                      apr_array_header_t **inherited_props,
                      svn_fs_root_t *root,
                      const char *path,
                      apr_pool_t *result_pool,
                      apr_pool_t *scratch_pool);

New APIs and Public Structs

/** A structure to represent inherited properties.
 *
 * @since New in 1.8.
 */
typedef struct svn_prop_inherited_item_t
{
  /** The absolute working copy path, relative filesystem path, or URL from
   * which the properties in @a prop_hash are inherited. */
  const char *path_or_url;

  /** A hash of (const char *) inherited property names, and (svn_string_t *)
   * property values. */
  apr_hash_t *prop_hash;
} svn_prop_inherited_item_t;

/**
 * Set @a *inherited_props to a depth-first ordered array of
 * #svn_prop_inherited_item_t * structures representing the properties
 * inherited by @a path at @a revision (or the 'head' revision if
 * @a revision is @c SVN_INVALID_REVNUM). Interpret @a path relative to
 * the URL in @a session. Use @a pool for all allocations. If no
 * inheritable properties are found, then set @a *inherited_props to
 * an empty array.
 *
 * Allocated @a *inherited_props in @a result_pool, use @a scratch_pool
 * for temporary allocations.
 *
 * @since New in 1.8.
 */
svn_error_t *
svn_ra_get_inherited_props(svn_ra_session_t *session,
                           apr_array_header_t **inherited_props,
                           const char *path,
                           svn_revnum_t revision,
                           apr_pool_t *result_pool,
                           apr_pool_t *scratch_pool);
* Return a deep copy of @a inherited_prop, allocated in @a result_pool.
* Use @a scratch_pool for temporary allocations.
* @since New in 1.8.
*/
svn_prop_inherited_item_t *
svn_prop_inherited_item_dup(const svn_prop_inherited_item_t *inherited_prop,
apr_pool_t *result_pool,
apr_pool_t *scratch_pool);

/**
* Return a deep copy the array @a prop_array of svn_prop_inherited_item_t *
* items in @a result_pool. Use @a scratch_pool for temporary allocations.
* @since New in 1.8.
*/
apr_array_header_t *
svn_prop_inherited_array_dup(const apr_array_header_t *prop_array,
apr_pool_t *result_pool,
apr_pool_t *scratch_pool)

/**
* Set @a *inherited_values to a depth-first ordered array of 
* #svn_prop_inherited_item_t * structures (the path_or_url members of which 
* are relative filesystem paths) representing the properties inherited by 
* @a path at @a revision in @a repos. If no properties are inherited, then 
* set @a *inherited_values to an empty array.
* @since New in 1.8.
*/
svn_error_t *
svn_repos_fs_get_inherited_props(apr_array_header_t **inherited_props,
svn_repos_t *repos,
const char *path,
svn_revnum_t revision,
svn_repos_authz_func_t authz_read_func,
void *authz_read_baton,
apr_pool_t *result_pool,
apr_pool_t *scratch_pool);

New Capabilities

/**
* The capability to get inherited properties.
* @since New in 1.8.
*/
#define SVN_RA_CAPABILITY_INHERITED_PROPS "inherited-props"
Drawbacks

The suggested design above has some known drawbacks and trade-offs:

- [JAF] I note that there is no ability to specify that an inheritable property should be deleted rather than overridden with a new value. Thus we can't use absent/present semantics similar to 'svn:needs-lock'. If we set an inheritable equivalent of 'svn:needs-lock = ""' on a parent dir, there is no way to designate a particular child node as explicitly removing that inherited property. That's probably fine – we can just avoid using the absent/present semantics for that kind of purpose – but it deserves to be mentioned.

1. Generic inherited properties also differ from mergeinfo in that the property value a child inherits from a parent is not based on the path-difference between the parent and child. The exact property value on the parent is the value the child will inherit.
2. See args to svn_client_proplist4 and svn_client_propget5 in the new APIs section. These are the primary APIs for getting inherited props from the WC.
3. svn_opt_revision_unspecified or svn_opt_revision_working.
4. The working tree plus property and text changes.