

The Binder

This document is confidential and may not be distributed without the permission of Be Inc.

copyright © 2000 Be Incorporated

The Binder

Introduction to the Binder

Description: BeIA global namespace; repository for system and user settings.

The Binder is a namespace tree that lets programs expose data and functionality to the rest of the system. The Binder's most important role is to expose system and user settings to programs, modules, and (local) HTML pages.

BeIA automatically loads the Binder tree on startup. The tree is statically defined in an XML file, and can be accessed dynamically through C++ code, through JavaScript code, or through the **binder** shell tool. Although C++ access is the most robust, JavaScript access coupled with modifying the XML definition should be sufficient for most vendors. Access through the **binder** tool is provided for testing and debugging.

This document looks at the Binder tree architecture and takes a brief look at the default Binder definition, particularly as it interests the vendor. The other Binder documents are:

- JavaScript access is described in *Binder JavaScript*.
- The C++ interface is <<forthcoming>>.
- The Binder XML file syntax is given in *The Binder XML Format*.
- The complete layout of the BeIA-defined Binder tree is in *The Binder Tree*.
- The **binder** tool is described in *The binder Program*.

1 Binder Architecture and Access

The binder tree is organized into branches, or **nodes**; each node contains named **properties**. A property is a name/value pair:

- The name is unique within that node.
- The value can be a simple type (a string or a number) or it can be another node object (that contains properties and other nodes, and so on).

Binder nodes can be nested to 64 levels, although it's anticipated that even in the most full-figured configuration, nesting won't need to be any deeper than ten or fifteen levels.

Note: Keep in mind that every node *is* a property. In general, features that apply to properties apply to nodes as well. Exceptions to this rule are pointed out in the documentation.

2 The XML Binder Description File

At boot time, the system builds the Binder tree by reading the XML file `/boot/home/config/settings/binder/root`. The file contains a persistent, human-readable depiction of the Binder's nodes and properties (with values). Here's a snippet from the file:

```
<object name="service" perms="rmd">

  <object name="audio" perms="r">
    <prototype perms="rw" stacking="true"/>
  </object>

  <object name="locale" perms="r">
    <prototype perms="rw" stacking="true"/>
    <string name="language">en</string>
  </object>
  ...
</object>
```

This excerpt creates the `service` node and its `audio` and `locale` subnodes. The `locale` node contains the `language` property which is set to the string value “en”.

The complete Binder XML syntax is in *The Binder XML Format*. The following sections give a brief overview of the XML file definition provided by BeIA, particularly as it interests the vendor.

2.1 The user-record.skel File

If you look in `/boot/home/config/settings/binder/`, you'll see, in addition to `root`, a file called `user-record.skel`. This is an XML Binder description that's copied into each user account as the accounts are created. This topic is discussed in greater detail in “**3.1.1** The User Account Template.”

Each user account is represented in a separate file. Management of the user account files is taken care of by a “user node” Binder plug-in.

2.2 Referring to Other Files

A property declaration in the `root` file can use the `storage` attribute to refer to some other file for its description:

```
<object name="vendor" perms="r">
  <prototype storage="/boot/binder/service/vendor" perms="rwcdm"
stacking="true"/>
</object>
```

This declares that the `vendor` node is defined in the `/boot/binder/service/vendor` file. When the Binder tree is initially created, this file is used to populate the `vendor` node. (The file doesn't exist by default—it's up to the vendor to create it.)

2.3 File Synchronization

Changes that are made to the Binder tree while a device is running are written to the appropriate storage file. For example, when the user changes a setting, the new value of that setting is written to the user's account file.

File-synchronization only applies to non-root files—the root XML file is never rewritten. To modify the root file on a user's device (once the device has been deployed) the vendor must replace it through a remote update to the device.

2.4 Vendor Files

The vendor is invited to augment the default Binder definition by creating or modifying these files:

- `/boot/binder/service/vendor`. As mentioned above, this is the file that represents the `beos.binder.service.vendor` node. The vendor is invited to define system-wide settings in this file.
- `/boot/home/config/settings/binder/user-record.skel`. This file was also described earlier. The file contains an empty vendor node declaration. The vendor is invited to create default user settings by filling out this node.

3 Predefined Binder Nodes

The root node contains three “subroot” nodes: `user`, `service`, and `application`. The following sections look briefly at the features of the three subroots. A map of the entire predefined Binder tree is given in *The Binder*.

3.1 The user Subroot

The `beos.binder.user` subroot contains a node for every user account, each of which contains user-specific settings and data—language, locale, bookmarks, email info, and so on.

3.1.1 The User Account Template

BeIA defines a template for new user accounts; the template is defined in the XML file `/boot/home/config/settings/binder/user-record.skel`. When a new user account is added to the system, the `user-record.skel` definition is automatically copied into the account. Subsequent changes to `user-record.skel` are not propagated into existing accounts.

3.1.2 Adding Vendor-defined Nodes and Properties

If, as a vendor, you want to supply your own user settings, you should add your nodes and properties to the vendor node in the `user-record.skel` file. How nodes and properties are organized within the vendor node is up to the vendor.

3.1.3 User Account Files

The data for each user account is stored as a `user-record` XML file in a subdirectory of the `/boot/binder/user` directory. The subdirectory is given the user’s name; for example, if a device has accounts for adam, betty, and charlie, the following XML files are created to store their settings:

```
/boot/binder/user/adam/user-record
/boot/binder/user/betty/user-record
/boot/binder/user/charlie/user-record
```

Freshly created, a `user-record` file is simply a copy of the template defined in the `/boot/home/config/settings/binder/user-record.skel`. As a user modifies his or her settings (language, email accounts, bookmarks), the corresponding user account file is automatically and immediately updated.

Although you can create a user account simply by creating a new subdirectory of `/boot/binder/user` and copying the desired `user-record` file into it, it's expected that user accounts will be managed dynamically, through the JavaScript calls explained in “**3.1.5** Adding and Removing User Accounts.”

By default, the system creates a system user account that's used on single-user systems:

```
/boot/binder/user/system/user-record
```

3.1.4 The Current User

The current user account (i.e. the account for the user that's logged into the device) is represented by the tilde (“~”) node in `user`, thus:

```
var currentUser = beos.binder.user[ "~" ];
```

As another example, here's how you retrieve the `vendor` node for the current user:

```
var vendorNode = beos.binder.user[ "~" ].vendor;
```

To set the current user, you do this:

```
beos.binder.user[ "~" ] = "Adam";
```

If you set the current user to `undefined`, the current user is set to the `system` account.

Note that the “~” node is a pointer to the node of the current user—the “~” account doesn't exist as `user-record` file.

3.1.5 Adding and Removing User Accounts

To create a new user account dynamically and add it to the `user` node, pass the account name to the node's `addUser()` function. For example:

```
beos.binder.users.addUser( "Adam" );
```

This creates the node `beos.binder.users.Adam` that contains properties that are copied from the `user-record.skel` file.

To delete a user account:

```
beos.binder.users.deleteUser( "Adam" );
```

You can't delete the `system` or “~” accounts.

3.2 The service Subroot

The `beos.binder.service` subroot provides system-wide information (device ID numbers, network configuration information, etc.), and software capabilities (lists of supported languages, printers, fonts, etc.). As examples, the ISP's telephone number is given in the `service.network` node; the various resolutions that a printer can handle for a given page are listed in the `service.printing` node.

3.2.1 The service.vendor Node

The `service` subroot contains a `vendor` node in which vendors can install their own machine-specific settings:

```
var vendorNode = beos.binder.service.vendor;
```

As with the `user.system.vendor` node, data organization within the `service.vendor` node is up to the vendor.

To modify the `user.system.vendor` node, edit the definition in the `/boot/home/config/settings/binder/root` file.

3.3 The application Subroot

The `beos.binder.application` subroot is used by applications (or plug-ins) to store app-specific data. An application registers itself within `application` by creating a node that's named for the application's MIME signature. For example, this node...

```
beos.binder.application.x-vnd-MoviePlayer
```

...would contain data for the (fictitious) plug-in with the MIME signature "application/x-vnd-MoviePlayer".

It's anticipated that new `application` nodes will only be created programmatically (through C++), when a new application or plug-in is first launched.

The Binder

Binder JavaScript

This document provides a general overview of how you use JavaScript to access the Binder tree. Many of the functions mentioned in this document are described in greater detail in the BeIA JavaScript reference documentation.

1 Identifying a Property

The Binder tree is accessed through the global `beos.binder` JavaScript object; this is the “root” of the Binder tree. To get to a specific property in the Binder, you add, to `beos.binder`, the names of the nodes that lead to that property, dot-delimiting each node. For example, this “node path”:

```
beos.binder.service.locale.language
```

...refers to the `language` property that’s inside the `locale` node that’s inside the `service` node.

You can also identify a property by indexing the name off of the parent node:

```
beos.binder.service.locale["language"]
```

1.1 Iterating over a Node’s Properties

A node lets you iterate over its properties. To do this, you treat the node as an array and ask for its n ’th property. Use the `length` property to test for the end of the (iterable) property list. The following example builds a list of printers:

```
var n;
var printers = beos.binder.service.printing;

for ( n=0; n < printers.length; n++ )
    document.writeln("<LI>" + printers[n].name);
```

An object that’s returned through iteration is a name/value pair that must be “decoded” through the `name` and `value` properties:

```
printers[n].name
printers[n].value
```

When accessing a property through iteration you *must* use `value` to get its value—even if the property is a node. For example, to tell the n ’th printer (which *is* a node) to print, you would do this:

```
printers[n].value.Print()
```

Since `printers[n]` was retrieved through iteration, leaving out the `value` keyword would be an error.

1.2 Functional Properties

Some properties are functions. To pass arguments to a functional property, put the arguments in parenthesis:

```
beos.binder.service.printing.Select("PrinterName")
```

Functional properties can act as nodes:

```
beos.binder.service.printing.Selected().Print()
```

2 Property Permission

There are five permission bits associated with each property (including nodes): read, write, create, delete, and (remote) mount. The permissions are defined in the XML Binder file; new properties (created dynamically by JavaScript) inherit their permissions from a `prototype` property declaration in the XML file.

You can't set the permission bits of an existing property through JavaScript, nor can you query for a property's permissions.

For more on property permissions, see `perms` in “The Binder XML Format” section.

3 Property Values

3.1 Getting a Property Value

A reference to a property returns its value:

```
if (beos.binder.service.locale.language == "en")
    ...
```

Note that for simple-valued properties, value-retrieval retrieves the value *only*—it doesn't retrieve the object that represents the property. This is a subtle point that's particularly important when you're using JavaScript variables: When you retrieve the value of a simple-valued property and stuff it into a JavaScript variable, as shown here, the variable is not “live”—a change to the property in the Binder does not update the value of the variable. If you need a live value, create a variable that points to the property's node, and then reference the property off of that node variable; for example:

```
var localeNode = beos.binder.service.locale;

if (localeNode.language == "en")
    ...
```

Setting the value of a simple-property through a JavaScript variable abides by the same restriction, as explained in “**3.3** Setting a Property Value.”

You can't get the value of a property that's read-protected.

3.2 Testing a Property Value's Data Type

To ask for the data type of a property's value, you call `typeof()` on the value returned by `valueOf()`:

```
typeof(beos.binder.service.locale.language.valueOf())
```

3.3 Setting a Property Value

You set a Binder property value through the assignment operator (“=”). For example:

```
beos.binder.service.audio.speaker.Mute = 0
```

The **Sounds** preference panel would use a line like this to unmute the internal speaker.

Keep in mind that if you're using JavaScript variables to represent a property's value, you can't simply set the value of the variable—the change won't be written back to the Binder. For example, consider these two versions of the speaker-muting code:

```
/* DO THIS: This works because "speakerNode" represents Mute's node: */
var speakerNode = beos.binder.service.audio.speaker;
speakerNode.Mute = 0;

/* DON'T DO THIS: It simply sets the value of the JavaScript variable "muteProp";
 * it doesn't actually set the value of the Mute property itself.
 */
var muteProp = beos.binder.service.audio.speaker.Mute;
muteProp = 0;
```

You can't set the value of a property that's write-protected.

4 Monitoring Properties

Binder lets you monitor changes to the values of properties. To monitor changes to a specific property, call `observe()` on the property:

```
beos.binder.service.locale.language.observe("watchPropFunc", "userData");
```

This registers `watchPropFunc`, a JavaScript that you create yourself, as the callback function that's called whenever the value of language changes. The second argument, `userData`, is arbitrary data that's passed back as an argument to the callback function.

To monitor all the properties in a node, call `observeContents()` on the node:

```
beos.binder.service.locale.observeContents("watchNodeFunc", "userData");
```

For more information on these functions, see `beos.binder` in the *BeIA JavaScript* chapter.

5 Creating a Property

To create a new property from JavaScript, simply set its value. If the property doesn't exist, it's automatically created and added to the referenced Binder node. For example, to add a (non-existent) "OldName" property to the `service.vendor` node, you would set the property's value:

```
beos.binder.service.vendor[OldName] = Ezzo;
```

The permission prototype for the node you're trying to add the property to must be writable and creatable (see the prototype documentation in "The Binder XML Format" section). Furthermore, the new property inherits the prototype's permissions, with one modification: All JavaScript-created properties are deletable (the "d" bit is set).

6 Deleting a Property

You can delete a property (i.e. remove it from the Binder) by setting its value to `undefined`, given that the property is deletable. For example, the following line removes *SomeProp*:

```
beos.binder.service.vendor.SomeProp = undefined
```

If the property *is* deletable, the value is set to undefined.

7 Creating a Node

To create a new Binder node, you first instantiate a Binder node template, and then add the instantiation to an existing Binder node. The available templates are defined as *skel* elements in the XML file that was used to create the Binder tree.

For example, let's say the XML file defines a *skel* called *toys* in the *beos.binder.vendor* node. To instantiate the *skel*, you reference off of the *beos.binder.vendor* node through the expression “+toys”:

```
var toysNode = beos.binder.vendor["+toys"];
```

(The + syntax is required when you're referencing a *skel*—it tells the Binder that the referenced object must be created first.)

toysNode is a real node object that can be modified. For example, if the *skel* definition declares a *Bells* property, you can set the value of that property as you would any other:

```
toysNode.Bells = 5;
```

If you want your new node to be persistent (i.e. to implant it in the Binder tree), you have to add it to a Binder node. Here we add our new node into the *beos.binder.vendor* node; we call the new node “toy1”:

```
beos.binder.vendor["toy1"] = toysNode;
```

At this point *toysNode* and *beos.binder.vendor.toy1* are the same object—a change to one will affect the other.

Typically, and as shown above, you add your new node back into the node that defined the *skel*. But this isn't a requirement—you can add the new node to any node that accepts new properties. For example, you could instantiate another *toys* *skel* and add it to the *toy1* node:

```
var subtoysNode = beos.binder.vendor["+toys"];
subtoysNode.Bells = 10;
beos.binder.vendor.toy1["subToy1"] = subtoysNode;
```

For more information, see *skel* in *The Binder XML Format*.

8 Property Objects and Reference Counting

When a Binder property is accessed, an object (in memory) is created by the system to represent it. Each node in the node path to the property is represented by a separate object. An object persists only if there's a reference to it (such as a variable). Subsequent references to the same properties are handed the previously created objects. References to a given object are counted; when the reference count goes to 0, the object is destroyed.

The persistence of property objects can be used to your advantage: By caching and reusing references to a property object, you can avoid recreating the same objects over and over. For example, the following code is extremely inefficient because each line creates (and then immediately destroys) the same set of objects:

```
if ( beos.binder.service.web.cache.size > 3000000)
    beos.binder.service.web.cache.size =
```

```
beos.binder.service.web.cache.size *  
beos.binder.service.web.cache.reduce_to;
```

The version below is much more efficient:

```
var webCache = beos.binder.service.web.cache;  
  
if ( webCache.size > 3000000)  
    webCache.size = webCache.size * webCache.reduce_to;
```

Note that this example *can't* be further reduced by creating a variable that refers to `webCache.size` and then setting the value of that variable. As explained earlier, setting the value of a property must be done in reference to the property's node—you can't simply set the value of a variable that directly represents the property.

The Binder

The Binder XML Format

The system builds the Binder tree at boot time by reading the XML file `/boot/home/config/settings/binder/root`. The file contains a persistent, human-readable depiction of the Binder's nodes and properties (with values). The document describes the syntax of the XML file. A separate document, *The Binder*, looks at the `root` file itself.

This document assumes you understand standard XML 1.0 syntax. The specification for XML 1.0 can be found at

<http://www.w3.org/TR/REC-xml>

1 Files

As mentioned above, the default Binder tree is read from the `/boot/home/config/settings/binder/root` file. This file may refer to other Binder XML files (**Binder storage** files) or plug-ins (**Binder handlers**) for the definitions of certain nodes. All Binder XML files have the same structure and vocabulary.

1.1 Synchronization with the Binder

The `root` Binder file is never rewritten by the system. The only way to modify the `root` file is through an update to the user's device. Thus, you should take care to define a robust and intelligent `root` file organization (in general, the default `root` file provided by Be should be sufficient).

Storage files, on the other hand, are automatically synchronized with the state of the Binder. Any (accepted) change to a property that was originally read from a Binder storage file is written back to the storage file by the system. If you want to be able to update the properties in a node and then store the new values across system reboots, you should store your node in a Binder storage file; see the [object](#) element for details.

Files that are created and managed by a Binder handlers are updated according to the handler. For example, the “user node” handler, which creates a user file for each user account, updates the appropriate file as the user adds bookmarks, changes his password, sets his email signature, and so on.

2 File Structure

The overall structure of the Binder XML file looks like this:

```
<?xml version="1.0"?>
<binder-object>

  < element >
  < element >
  < element >
  ...
</binder-object>
```

Everything between the open and close `binder-object` elements makes up the default Binder definition. Most of the contents of the file is a series of nested `object` elements. Each `object` represents a node; `objects` contains property-defining elements such as `string`, `number`, `undefined`, and `object` (a nested node). Each `object` should also contain a `prototype` element that sets the permission bits for new properties that are created dynamically. A property can also set permissions for itself.

A section of the definition looks something like this:

```
<object name="nodeA" >
  <prototype perms="rwcd" /> <!-- Permissions for new properties of nodeA -->
  <object name="nodeA1" perms="r">
    <prototype perms="rw" /> <!-- Permissions for new properties of nodeA1 -->
    <string name="propertyA1a" perms="r">value1</string>
    <number name="propertyA1b" perms="rd">5</string>
    <undefined name="propertyA1c" />
  </object>

  <object name="nodeA2"... >
    <prototype perms="r" />
    <string name="propertyA2a" perms="rd">value2</object>
    <number name="propertyA2b" >6</object>
    <undefined name="propertyA2c" />
  </object>
</object>

<object name="nodeB" storage="boot/binder/nodeB" />
```

The specifications of the elements are given in the next section.

3 Elements

There are eleven Binder elements:

- `binder-object` delimits the scope of the Binder XML definition.
- `object` and `mountpoint` define nodes.
- `skel` defines a template for a node.
- `string`, `number`, and `undefined` define properties. (In the spirit of completeness, we should mention that since nodes are properties, `object` and `mountpoint` define properties as well.)
- `prototype` defines a set of attributes that are copied into a node's properties.
- `overlay` and `inherit` tell a node takes on the properties of some other node.
- `include` tells the XML file reader to include the contents of some other XML file.

A number of the elements take the `perms` attribute, which sets the permissions for the entity created by the element. There are five permission bits, represented by the characters 'r', 'w', 'c', 'd', and 'm'. The

meanings of these permissions depends on the element; each element that supports the attribute explains their meanings.

binder-object

- Binder hierarchy declaration

```
< binder-object >
  elements
< /binder-object >
```

The `binder-object` element declares that the contents should be used to create a Binder hierarchy. The `binder-object` in the `/boot/home/config/settings/binder/root` file defines the hierarchy at the root of the binder (in JavaScript parlance, it's added to the `beos.binder` object). When read from other XML files that are referred to from the `root` file, the hierarchy is placed in the node at which the file is read (see the `storage` argument for details).

include

- Include the contents of some other XML file.

```
< include attributes / >
```

The `include` element lets you read and include the contents of some other XML file. Including a file is relevant only when the Binder is being constructed—the Binder doesn't write back into the included file. The element takes a single attribute:

- `storage`. The absolute pathname of the included XML file.

inherit

```
< inherit handler = handlerName >
  handlerArguments
< /inherit >
```

`inherit` declares that the node that contains this statement will take on the properties in the node defined by the named handler. You can pass whitespace delimited arguments to the handler through the `handlerArguments` data. If the handler defines a property that already exists in this node, the node's property takes precedence.

mountpoint

```
< mountpoint attributes / >
```

`mountpoint` defines a placeholder for a node that's "mounted" by a remote application (i.e. an application that's not running in Binder's address space). The element takes the following attributes:

- `name`. The name of the mountpoint.
- `perms`. Permission bits. If the element doesn't include a `perms` attribute, the property inherits its permissions from the node's `prototype` element.

number

- Creates a number-valued property.

```
< number attributes >
    value
< /number >
```

`number` creates a number-valued property. In all other details, it's the same as `string`.

object

- Binder node declaration

```
< object attributes >
    elements
< /object >
```

The `object` element creates a new Binder node. It takes the following attributes:

- `name`. The string name of the property. The name must be unique within the parent node.
- `handler`. The name of the Binder add-on (or plug-in) that's loaded and executed to handle the creation and management of this node. The named add-ons is searched for in these directories (in this order): `/boot/beos/~/.add-ons/binder`, `boot/beos/common/add-ons/binder`, `/boot/beos/system/add-ons/binder`.
- `storage`. The full path of the file that the node's XML representation is written to. By convention, the pathname emulates the node path to the node, but with `/boot` replacing `beos`. For example, the `beos.binder.service.vendor` node is stored in the `/boot/binder/service/vendor` file. When the Binder is shut down (and at other convenient moments), the contents of the node are written to the storage file; when the Binder is restarted, the node is populated with the contents of the file.
- `perms`. Permission bits. If the element doesn't include a `perms` attribute, the property inherits its permissions from the node's `prototype` element.

`handler` and `storage` are optional. A node can have both a handler and a storage file.

The `object` element can contain any of the other elements except `binder-object`. Of particular importance is the `prototype` element; if you want to be able to add properties dynamically to your node (through JavaScript), the object that represents the node must contain a `prototype` element. New properties that are added to the node "inherit" the prototype's permissions.

Typically, an `object` element contains one `prototype` element followed by a series of elements that define the node's properties (`string`, `number`, `undefined`, and `object`).

overlay

```
< overlay handler = handlerName >
    handlerArguments
< /overlay >
```

This is the same as `inherit`, except that in the case of property-name collisions, the handler wins.

prototype

- Template for dynamically-created properties.

```
< prototype attributes />
```

`prototype` is used inside an `object` element to declare attributes that apply or are copied into the node's other properties. It takes the following attributes:

- `stacking`. << forthcoming >>
- `ordered`. << forthcoming >>
- `perms`. The permission bits are copied into dynamically created properties, as well as into any existing (i.e. XML-defined) properties that don't have explicit `perms` attributes of their own. To be able add new properties to this `prototype`'s node from JavaScript, the `perms` attribute must include the "c" and "w" bits.

skel

- Template for dynamically-created nodes.

```
< skel attributes >
  elements
< /skel >
```

A `skel` element is a template for a dynamically created node. The elements in the `skel` object are the same as those found in `object`: Typically, a `skel` contains a `prototype`, followed by elements that define the node's properties.

`skel` takes the same attributes as `object`.

string

- Declares a string-valued property

```
< string attributes >
  value
< /string >
```

`string` creates a string-valued property. The property's value is set to `value`; without a `value`, the property is set to `undefined`. The element takes these arguments:

- `name`. The name of the property.
- `perms`. Permission bits. If the element doesn't include a `perms` attribute, the property inherits its permissions from the node's `prototype` element.

undefined

- Reserves a property name without giving it a value.

```
< undefined attributes / >
```

The `undefined` element lets you "reserve" a property name without giving it a value, and without declaring the value's type. `undefined` takes the following attributes:

- **name**. The name of the property.
- **perms**. Permission bits. If the element doesn't include a **perms** attribute, the property inherits its permissions from the node's **prototype** element.

4 Attributes

name

- Provides a name for an element.

```
< element name = name ... >
```

ordered

- Declares

```
< element ordered = "true" | "false" ... >
```

perms

```
< element perms = "rwcdm" ... >
```

The **perms** argument declares whether a Binder node, property, or property prototype can be read, written, deleted, and so on. There are five permission bits, represented by the characters listed below. The value of **perms** is a concatenation of the appropriate characters.

Char	Meaning
r	Read: The value of the property can be read. For a node, it means the names of the node's properties can be retrieved and referenced.
w	Write: The value of the property can be modified. Writable properties can change type. For example, if you set the value of a (writable) node to be a string, the node will “lose” its property list and “become” a string.
c	Create: The property is creatable. This is primarily significant for prototype elements. If a prototype doesn't include the “c” bit, you won't be able to add new properties to the prototype 's node.
d	Delete: The property can be removed from the Binder tree. When you set a (deletable) property's value to the JavaScript value undefined , the property is deleted. If it's not deletable, the property's value is set to undefined . Dynamically created properties always have the delete bit set.
m	Mount: This applies to nodes only. A mountable node can be “mounted” from an application that's running in some other address space (i.e. other than the address space that Binder runs in).

The Binder

The Binder Tree

Description: Default Binder tree content.

API Type: XML definition that can be interpreted as JavaScript or C++ objects.

Declared in: `/boot/home/config/settings/binder/root` et al

This document describes the contents (nodes and properties) of the Binder tree as defined by Be.

When a BeIA device is launched, the Binder tree is populated by reading the node and property definitions in the `/boot/home/config/settings/binder/root` XML file. This file refers to other XML files (user account files, network settings, vendor-specific files, and so on) to complete the Binder tree population. The Binder's nodes and properties are most often interpreted as JavaScript objects; you can also explore the Binder tree through the C++ Binder classes.

The major sections below describe the contents of each of the XML files that BeIA uses to populate the Binder tree.

1 The Root File (The binder Node)

Declared in: `/boot/home/config/settings/binder/root`

Permissions: node: *none* prototype: *r*

Description: Defines the binder node (`beos.binder` in JavaScript), declares the three major Binder nodes (`application`, `user`, and `service`), and populates most of the `service` node.

Nodes: `application`

Used by applications to store app-specific data.

`service`

System-wide settings and lists of device capabilities.

`user`

User accounts.

1.1 application

Permissions: node: *r* prototype: *rmd*

Description: The `application` node is used by applications (or plug-ins) that store app-specific data. The default definition is empty.

1.2 service

Permissions: node: rmd prototype: *none*

Description: The `service` node provides system-wide information (device ID numbers, network configuration information, etc.), and software capabilities (lists of supported languages, printers, fonts, etc.).

Nodes:

- `audio`
Audio information.
- `auto_mounter`
Mounted file systems.
- `email`
Global email settings and functions.
- `indicators`
Interface to the hardware status indicators (LEDs and the like).
- `locale`
Location-specific information.
- `mca`
MAP client agent settings.
- `updater`
Remote update settings.
- `network`
Network settings.
- `printing`
List of available printers, and print commands.
- `softkeyboard`
On-screen keyboard info and access.
- `system`
Miscellaneous device information (ID, software build date, etc.)
- `vendor`
Reserved for use by the vendor.
- `video`
Video information
- `web`
Web browser information.

1.2.1 service.audio Currently unused

Permissions: node: r prototype: rw stacking: true

Description: Audio information.

1.2.2 service.auto_mounter

See “2 Vendor-specific Extensions to the Root File”.

1.2.3 service.locale

Permissions: node: r prototype: rw stacking: true

Description: Location-specific information.

Properties: language

The two-character ISO representation of the device’s natural language. Set to “en” by default.

1.2.4 service.network

Permissions: node: r prototype: none

Description: Network configuration information.

Note: Documentation for the `network` node is currently being developed. Preliminary documentation is provided in *The service.network Node*.

1.2.5 service.printing

Permissions: node: r prototype: rm

Description: Each node in `service.printing` represents a currently available printer.

Note: The `service.printing` node is managed by a Binder plug-in; the root file doesn’t populate the node.

Properties: Selected

The name of the currently selected printer. This will be the name of one of `service.printing`’s nodes.

Select (*printer*)

Sets the current printer to `printer`, which must be the name of one of `service.printing`’s nodes.

Nodes: <printer>

Represents an available printer.

1.2.5.1 service.printing.<printer>

Permissions: NA

Description: Each *printer* node represents a currently available printer. The information for the printer is provided by the driver for that printer.

Properties: Cancel

Cancels a print job.

GetOrientation

Returns the paper's layout orientation; either "portrait" or "landscape".

InkLevel

Number [0, 100] that represents the amount of ink that's in the printer. 0 means empty; 100 means full. `InkLevel` doesn't appear in the node until the printer has actually printed.

PrettyName

The UI-acceptable form of the printer's name.

Print

Commands the printer to print.

SelectedPaperFormat

Returns the name of the currently selected paper format ("Letter", "Legal", "Tabloid", etc., as defined by the printer driver). This will be the name of one of the nodes in `printing.<printer>.papers_formats`.

SelectedPaperType

Returns the name of the currently selected paper type ("Glossy", "Matte", "Normal", etc., as defined by the printer driver). This will be the name of one of the nodes in `printing.<printer>.papers`.

SelectedResolution

Returns the name of the currently selected resolution ("Fine", "Normal", "Course", etc., as defined by the printer driver). This will be the name of one of the resolution properties in `printing.<printer>.papers.<paper>`. The resolution is set as a side-effect of `SelectPrintMode()`.

SelectPaperFormat (*format*)

Sets the current paper format. *format* must be one of the properties in `printing.<printer>.papers_formats`.

SelectPrintMode (*mode*)

Sets the current "printing mode". This is a combination of a paper type and a paper resolution. You get a *mode* suitable for use here through one of the resolution properties in `printing.<printer>.papers.<paper>`. In other words, you set the mode by passing in a resolution. Because resolutions are grouped by paper type, the paper type implied by the resolution is unambiguous.

SetOrientation (*orientation*)

Sets the paper's layout orientation; *orientation* must be either "portrait" or "landscape".

Status

Returns a printing status code. This is a string that looks like one of the Be error codes ("B_OK", "B_PRINTER_COVER_OPEN", "B_UNKNOWN", etc.). The status is "B_UNKNOWN" until the printer is actually used.

Nodes: `papers`
 List of paper types (nodes).
 `paper_formats`
 List of paper formats (nodes).

1.2.5.1.1 `service.printing.<name>.papers`

Permissions: *NA*

Description: Contains a list of paper types, where each type is represented by a separate node.

Nodes: `<paperType>`
 Each node represents a type of paper.

1.2.5.1.1.1 `service.printing.<name>.papers.<paperType>`

Permissions: *NA*

Description: Publishes a list of resolutions for this paper type.

Properties: `<resolution>`
 Each property names a resolution. In the interest of unambiguous UI, each resolution name should be unique across *all* paper types.

1.2.5.1.2 `service.printing.<name>.paper_formats`

1.2.6 `service.softkeyboard`

Note: Most of the node descriptions from here to the end of the document have yet to be converted to the style used in the top half of this document. The difference in formatting shouldn't be construed to signify anything meaningful about the nodes themselves

Node perms: `r`

Prototype perms: `rw`

`service.softkeyboard` contains properties that describe features of the on-screen keyboard:

- `repeat_delay`. The amount of time to wait, in microseconds, before repeating a key. 750000 by default.
- `repeat_rate`. The number of repeated key iterations per second. 13 by default.

1.2.7 service.system

Node perms: r
Prototype perms: r

service.system contains information about the operating system. Its properties are:

- `altq`. A boolean value: 1 if `alt+q` dumps the user into the desktop environment, and 0 if `alt+q` does nothing.
- `bootmode`. The environment the device boots into. One of “tools”, “validate”, “firstboot”, and “normal”. See “*Boot Mode*” in the *BeIA Support* book for more information.
- `build`. The date the operating system was built, given as “*yyyy.mm.dd*”.
- `device_type`. Defined by the device; see “**3** Device-specific Extensions to the Root File” for more information.
- `product`. The name of the operating system (“BeIA”).
- `version`. A string that identifies the version of the operating system.

1.2.8 service.system.bios_node

Provides information about the BIOS:

- `bios_date`.
- `bios_vendor`.
- `bios_version`.
- `device_id`.

1.2.9 service.updater

Node perms: r

Information used by the update mechanism.

1.2.10 service.vendor

Node perms: r
Prototype perms: rwcdm

The `service.vendor` node is reserved for the vendor. See “**2** Vendor-specific Extensions to the Root File” for more information.

1.2.11 service.video Currently unused

Node perms: r

Prototype perms: rw

The `service.video` node contains video information.

1.2.12 service.web

Permissions: node: r prototype: rw stacking: true

Storage: /boot/binder/service/web

Description: Information used by a Web browser to access the Web.

- Nodes:**
- cache
 - macros
 - navigator
 - proxy
 - security
 - state
-

1.2.12.1 service.web.cache

Permissions: node: *none* prototype: rw

Description: Web page cache size information.

Properties: size

Size of the browser cache, in bytes.

reduce_to

Factor by which the cache is reduced, if necessary.

1.2.12.2 service.web.macros

Prototype perms: rwcd

- RESOURCES. Pathname to the user interface files.
- SCRIPTS. Pathname to the cgi-bin scripts.

1.2.12.3 service.web.navigator

Prototype perms: rw

- user_agent.
- app_version.
- app_name.
- app_code_name.
- platform.

1.2.12.4 service.web.proxy

Prototype perms: rw

- http_server.
- http_port.

1.2.12.5 service.web.security

Prototype perms: r

service.web.security contains a list of servers that are given special access.

1.2.12.5.1 service.web.security.acl

Node perms: r

Prototype perms: rwcd

Each node in service.web.security.acl (Access Control List) describes a set of access permissions that are applied to a group of secure servers (as defined in “1.2.12.5.2 service.web.security.groups”). The types of access are listed as properties of the individual nodes; the value of an access property is either “grant” or “revoke”.

1.2.12.5.1.1 service.web.security.acl.custom_content

Prototype perms: rwcd

- custom_content. Granted.
- mail_workers. Granted.

1.2.12.5.1.2 service.web.security.acl.internet

Prototype perms: rwcd

- all. Granted.

1.2.12.5.1.3 service.web.security.acl.intranet

Prototype perms: rwcd

- custom_content. Granted.

1.2.12.5.1.4 service.web.security.acl.local_machine

Prototype perms: rwcd

- all. Granted.
- internet. Revoked.
- intranet. Revoked.

1.2.12.5.1.5 service.web.security.acl.mail

Prototype perms: rwcd

- custom_content. Granted.
- mail. Granted.
- mail_workers. Granted.

1.2.12.5.1.6 service.web.security.acl.mail_workers

Prototype perms: rwcd

- custom_content. Granted.
- mail. Granted.
- mail_workers. Granted.

1.2.12.5.2 service.web.security.groups

Node perms: r

Prototype perms: rwcd

service.web.security.groups contains a list of “secure” servers identified by URL. A server URL address can contain “*” as a wildcard, but use caution—use wildcards only at the leaf end of a URL.

- *URL*. The name of each property is a separate URL. The value of the property is the name of one of the Access Control List nodes (described in “**1.2.12.5.1** service.web.security.acl”).

1.2.12.6 service.web.state

1.3 user

The user node contains a list of user accounts. Creation and management of these accounts is handled by a Binder plug-in. Properties for a new account are copied from the user-record.skel file (see “**4** The User Account Template”).

2 Vendor-specific Extensions to the Root File

Declared in: `/boot/home/config/settings/binder/vendor-specific-root`

The `vendor-specific-root` file contains nodes and properties that are defined by (or for) the vendor. The file is overlaid on top of the Binder tree defined in `root`, and takes precedence in case of conflicting properties.

2.1 service

2.1.1 service.auto_mounter

`service.auto_mounter` provides access to the file-system volume mounter. It contains nodes that list the drives and the volumes that are currently attached to the system.

2.1.1.1 service.auto_mounter.drives

Node perms: `rwd`

The `drives` node contains a node for each drive attached to the system, as described below.

2.1.1.1.1 service.auto_mounter.drives.<name>

Each named node provides the following information about the drive it represents:

- `device.`
- `product.`
- `removable.`
- `type.`
- `vendor.`
- `version.`

2.1.1.2 service.auto_mounter.volumes

Node perms: `rwd`

The `volumes` node contains a node for each volume attached to the system, as described below.

2.1.1.2.1 service.auto_mounter.volumes.<name>

Each named node provides the following information about the volume it represents:

- `volume_name.`
- `vounted_at.`
- `device.`
- `block_size.`
- `total_blocks.`
- `free_blocks.`

- `removable`.
- `read_only`.

3 Device-specific Extensions to the Root File

Declared in: `/boot/home/config/settings/binder/device-specific-root`

The `device-specific-root` file contains information that pertains to the type or model of this device.

3.1 service

3.1.1 service.system

One property is added to the `service.system` node:

- `device_type`. A string that identifies this device's type.

4 The User Account Template

Declared in: `/boot/home/config/settings/binder/user-record.skel`

The `user-record.skel` file is a template that's used as the basis for user accounts. When a new user is added to the device, the system creates a subdirectory of `/boot/binder/user`, names the subdirectory after the new user, and then copies `user-record.skel` into the subdirectory (renaming the file `user-record`). For example, if a device has accounts for adam, betty, and charlie, the following XML files are created to store their settings:

```
/boot/binder/user/adam/user-record
/boot/binder/user/betty/user-record
/boot/binder/user/charlie/user-record
```

4.1 user

The `user` node defines these functions:

- `addUser(name)`. Adds a new user account by creating new `user-record` file in the directory `/boot/binder/user/<name>`.
- `deleteUser(name)`. Deletes the `name` user account by removing the `/boot/binder/user/<name>` directory.

4.1.1 user.vendor

Node perms: `r`

Prototype perms: `rwcd`

The `user.vendor` node contains information that's written by (or for) the vendor.

4.1.2 user.info

Node perms: rw

Prototype perms: rw

`user.info` contains user information:

- `FirstName`. The user's first name.
- `LastName`. The user's last name.

4.1.3 user.email

The user's email information.

4.1.3.1 user.email.account

Prototype perms: rw

Information about the user's "current" email account.

- `DisplayName`. Name that appears in the "From" field in messages that are written by this user.
- `Organization`. Optional email header information.
- `Email`. The user's email address.
- `ReplyTo`. The user's reply-to address.
- `Signature`. The default signature for this account.
- `Login`. Login name to the Imap server.
- `Password`. Password to the Imap server.
- `ImapServer`. Hostname or IP address of the Imap server.
- `ImapPort`. Imap port number; typically 43 (the default).
- `Smtpport`. SMTP port number; typically 25 (the default).
- `SmtppServer`. Hostname or IP address of the SMTP server.

4.1.3.2 user.email.signatures Currently unused

4.1.4 user.bookmarks

Node perms: rwcdm

Prototype perms: rwcdm

The user's browser bookmarks (or "favorites"). Each node is the name of a bookmark.

4.1.4.1 user.bookmarks.<name>

Node perms: rwcdm

Prototype perms: rwcdm

A bookmark description node contains these properties:

- `URL`. The URL of the bookmark.
- `isSecure`. A boolean value: 1 if this is a secure site, 0 if not.
- `isDeletable`. A boolean value: 1 if this bookmark can be deleted, 0 if not.

4.1.5 `user.cookies`

Node perms: <code>rwcdm</code>
Prototype perms: <code>rwcdm</code>

The user's cookie list. Each node is a single cookie.

4.1.5.1 `user.cookies.<name>`

Node perms: <code>rwcdm</code>
Prototype perms: <code>rwcdm</code>

A cookie description node contains these properties:

- `name`. The cookie's ID.
- `value`. The cookie's value.
- `domain`. The domain this cookie applies to.
- `path`. File system pathname where the cookie was created.
- `expiration`. Expiration date as a [time_t](#).

4.1.6 `user.addressbook`

Node perms: <code>rwcdm</code>
Prototype perms: <code>rwcdm</code>

The user's address book. Each node contains information for an entry.

4.1.6.1 `user.addressbook.<name>`

Node perms: <code>rwcdm</code>
Prototype perms: <code>rwcdm</code>

An addressbook entry contains these properties:

- `nickname`.
- `firstName`.
- `lastName`.
- `email`.

5 Node Map

The following list provides a quick reference to all the Binder nodes currently defined by Be.

application (*no nodes*)

service

- service.audio Currently unused
- service.automounter
 - service.automounter.drives
 - service.automounter.drives.<name>
 - service.automounter.volumes
 - service.automounter.volumes.<name>
- service.locale
- service.network
- service.printing
 - service.printing.<name>
 - service.printing.<name>.papers
 - service.printing.<name>.papers.<paperName>
 - service.printing.<name>.paper_formats
- service.softkeyboard
- service.system
 - service.system.bios_node
- service.updater
- service.vendor
- service.video Currently unused
- service.web
 - service.web.cache
 - service.web.macros
 - service.web.navigator
 - service.web.proxy
 - service.web.security
 - service.web.security.acl
 - service.web.security.acl.custom_content
 - service.web.security.acl.internet
 - service.web.security.acl.intranet
 - service.web.security.acl.local_machine
 - service.web.security.acl.mail
 - service.web.security.acl.mail_workers
 - service.web.security.groups
 - service.web.state

user

- user.vendor
- user.info
- user.email
 - user.email.account
 - user.email.signatures Currently unused
- user.bookmarks
 - user.bookmarks.<name>
- user.cookies
 - user.cookies.<name>
- user.addressbook
 - user.addressbook.<name>

The Binder

The service.network Node

Description: Contents of the `beos.binder.service.network` node.

API Type: Can be interpreted as JavaScript or C++.

Declared in: A Binder add-on.

This document describes the contents (nodes and properties) of the `service.network` branch of the Binder tree. The main Binder tree description is in *The Binder Tree* section. The network branch is broken out and described here because of its size.

Note: This information in this documentation is up-to-date, but the explanations, layout, and formats are preliminary.

1 Concepts

1.1 service/network

`service/network` is the top level of the network portion of the Binder tree:

```
'service/network' --> ''
|
+--'country_codes' -->
|
+--'profiles' -->
|
+--'control' -->
```

- `service/network/country_codes` is read-only backing store for all the modem country codes used by the unit's modem. This is pretty much only used by the UI, so most of you can ignore it.
- `service/network/profiles` is the backing store of network configuration data. It is read/write, but in practice no current state info is read from it.
- `service/network/control` is the object used to control networking and monitor status.

1.1.1 service/network/control

The `service/network/control` object is the most interesting bit (see the Reference section for the full deal):

```
'service/network/control' --> ''
|
+--'DNS' -->
|
```

```

+--'status' -->
|
| *'hostname' --> string('gpz.be.com')
| *'profile' --> string('dhcp')
|
+--'interfaces' -->
|
+--'route' -->

```

There are three functions not shown here:

service/network/control/adopt(const char *)

this function causes the net node to adopt and use a named profile from the backing store

service/network/control/up()

this function brings the current profile up

service/network/control/down()

this function brings the current profile down

1.1.1.1 service/network/control/DNS

service/network/control/DNS is an object that reports the current DNS settings.

1.1.1.2 service/network/control/status

service/network/control/status is an object reflecting the in-kernel state of the network interfaces and default route. It also reports the name of the current profile being used and the hostname. The status node is where all current state info should be read from/ observed/etc.

1.1.1.2.1 service/network/control/status/interfaces

service/network/control/status/interfaces is a read-only list of the interfaces as currently configured. Each interface under interfaces has a number of properties. Using the ppp interface as an example:

```

+--'status' -->
|
| +--'interfaces' -->
| |
| | +--'ppp0' -->
| | |
| | | *'address' --> string('10.1.1.1')
| | | *'bandwidth' --> number(56000.000000)
| | | *'hwaddr' --> string('unknown')
| | | *'index' --> number(4.000000)
| | | *'linkmsg' --> string('ok')
| | | *'linkstatus' --> string('disconnected')
| | | *'mask' --> string('255.0.0.0')
| | | *'status' --> string('down')
| | | *'type' --> string('ppp')
| |

```

- address is the current (or last) IP address of the interface.
- bandwidth is the number of bits/sec that the interface can be expected to achieve at max throughput.
- hwaddr is the hardware address of the interface (useful only on ethernet cards).
- index is the internal index number of the interface.
- linkmsg and linkstatus are (currently) PPP-specific items that reflect both the detailed current state of the device, and what happened.

- linkstatus will have values like connected, dialing, disconnected, etc.
- linkmsg will contain the "why" in case an error happens, with sample values like "no carrier", "busy", "no answer", etc.
- mask is the IP address mask in use.
- status is the current state of the interface. It has three possible values: up, down, or pending. A device may be pending while dialing, waiting for a DHCP lease, etc.
- type is the type of interface (in this case, ppp). Other values are pppoe, ethernet, or loopback.

1.1.1.2.2 service/network/control/status/route

service/network/control/status/route reports the current in-kernel default route.

Both the default route and each of the interfaces are updated asynchronously when things change in the kernel. They should be observed. You can poll them if you like, but observing them is much better for obvious reasons.

1.1.2 service/network/profiles

The service/network/profiles object is the backing store for all network configuration data. I'm not going to describe all the many parameters here, but I will give an overview of how it works.

```
'service/network/profiles' --> ''
| *'currentprofile' --> string('dhcp')
|
|--'custom' -->
|
|--'dhcp' -->
| |
| | |--'interfaces' -->
| | |
| | | |--'ethernet' -->
| | | |
| | | |--'loopback' -->
| | | |
| | | |--'ppp' -->
| | |
| |
| | |--'ppp' -->
| | |
| | | |--'interfaces' -->
| | | |
| | | | |--'ethernet' -->
| | | | |
| | | | |--'loopback' -->
| | | | |
| | | | |--'ppp' -->
| | | |
| | |
| | | |--'ppp800' -->
| | | |
| | | | |--'interfaces' -->
| | | | |
| | | | | |--'ethernet' -->
| | | | | |
| | | | | |--'loopback' -->
| | | | | |
| | | | | |--'ppp' -->
```

```

|
|--'pppoe' -->
|
|   |--'interfaces' -->
|   |
|   |   |--'ethernet' -->
|   |   |
|   |   |--'loopback' -->
|   |   |
|   |   |--'ppp' -->
|   |
|
|--'staticip' -->
|   |--'dns_domain' --> string('be.com')
|   |--'primary_dns' --> string('207.155.183.72')
|   |--'secondary_dns' --> string('206.173.119.72')
|   |
|   |--'interfaces' -->
|   |
|   |   |--'ethernet' -->
|   |   |
|   |   |--'loopback' -->
|   |   |
|   |   |--'ppp' -->
|   |
|
|--'route' -->

```

service/network/profiles/currentprofile contains the name of the profile to be used on boot by the node.

Ignore service/network/profiles/custom for now, it is not used.

service/network/profiles/dhcp, ppp, pppoe, ppp800, and staticip are what are known as Networking Profiles. Each Networking Profile contains configuration info for all devices on the system. Essentially each networking profile is a snapshot of the configuration for each type of use. Each contains an interfaces object, which is a list of all the interfaces on the device. For all devices so far, this list contains one ethernet interface, one loopback interface, and one ppp interface. Each of the interface objects in the list contains various properties used to configure that interface.

The staticip profile also contains backing store for the static IP DNS information and the static IP default route/gateway.

There are many properties, please see the attached sample files.

1.1.3 service/network/country_codes

The country_codes object is simple:

```

|--'country_codes' -->
|   |--'command' --> string('+gci=')
|   |
|   |--'codes' -->
|   |   |--'Argentina' --> string('07')
|   |
|
[...]
```

- country_codes/command is the command issued to the modem to use for the country codes.
- country_codes/codes contains a series of codes, one per region.

2 Reference

```

service/network/control' --> ''
|
|--'DNS' -->
|   *'domain' --> string('')
|   *'primary' --> string('207.155.183.72')
|   *'secondary' --> string('206.173.119.72')
|
|--'status' -->
|   *'hostname' --> string('gpz.be.com')
|   *'profile' --> string('dhcp')
|
|--'interfaces' -->
|   |--'/dev/net/tulip/0' -->
|       *'address' --> string('192.168.1.100')
|       *'bandwidth' --> number(10000000.000000)
|       *'hwaddr' --> string('00:c0:f0:37:63:ec')
|       *'index' --> number(1.000000)
|       *'mask' --> string('255.255.255.0')
|       *'status' --> string('up')
|       *'type' --> string('ethernet')
|   |--'loop0' -->
|       *'address' --> string('127.0.0.1')
|       *'bandwidth' --> number(10000000.000000)
|       *'hwaddr' --> string('zen-like')
|       *'index' --> number(2.000000)
|       *'mask' --> string('255.0.0.0')
|       *'status' --> string('up')
|       *'type' --> string('loopback')
|   |--'ppp0' -->
|       *'address' --> string('10.1.1.1')
|       *'bandwidth' --> number(56000.000000)
|       *'hwaddr' --> string('unknown')
|       *'index' --> number(4.000000)
|       *'linkmsg' --> string('ok')
|       *'linkstatus' --> string('disconnected')
|       *'mask' --> string('255.0.0.0')
|       *'status' --> string('down')
|       *'type' --> string('ppp')
|
|--'route' -->
|   *'address' --> string('192.168.1.1')
|   *'interface' --> string('/dev/net/tulip/0')
|
'service/network' --> ''
|
|--'country_codes' -->
|   *'command' --> string('+gci=')
|   |--'codes' -->
|       *'Argentina' --> string('07')
|       *'Australia' --> string('09')
|       *'Austria' --> string('0a')

```

```

*'Belgium' --> string('0f')
*'Boliva' --> string('14')
*'Brazil' --> string('16')
*'Bulgaria' --> string('1b')
*'Canada' --> string('20')
*'Chile' --> string('25')
*'Columbia' --> string('27')
*'Costa Rica' --> string('2b')
*'Cyprus' --> string('2d')
*'Czech Republic' --> string('2e')
*'Denmark' --> string('31')
*'Ecuador' --> string('35')
*'Finland' --> string('3c')
*'France' --> string('3d')
*'Germany' --> string('42')
*'Greece' --> string('46')
*'Guatemala' --> string('49')
*'Hong Kong' --> string('50')
*'Hungary' --> string('51')
*'Iceland' --> string('52')
*'India' --> string('53')
*'Indonesia' --> string('54')
*'Ireland' --> string('57')
*'Isreal' --> string('58')
*'Italy' --> string('59')
*'Japan' --> string('00')
*'Korea' --> string('61')
*'Liechtenstein' --> string('68')
*'Luxembourg' --> string('69')
*'Malaysia' --> string('6c')
*'Mexico' --> string('73')
*'Netherlands' --> string('7b')
*'New Zealand' --> string('7e')
*'Nicaragua' --> string('7f')
*'North America' --> string('b5')
*'Norway' --> string('82')
*'Pakistan' --> string('84')
*'Panama' --> string('85')
*'Paraguay' --> string('87')
*'People's Republic of China' --> string('26')
*'Peru' --> string('88')
*'Philippines' --> string('89')
*'Poland' --> string('8a')
*'Portugal' --> string('8b')
*'Puerto Rico' --> string('8c')
*'Russia' --> string('b8')
*'Saudi Arabia' --> string('98')
*'Singapore' --> string('9c')
*'South Africa' --> string('9f')
*'Spain' --> string('a0')
*'Sweden' --> string('a5')
*'Switzerland' --> string('a6')
*'Taiwan' --> string('b5')
*'Thailand' --> string('a9')
*'Turkey' --> string('ae')
*'United Kingdom' --> string('b4')
*'Uruguay' --> string('b7')
*'Venezuela' --> string('bb')
*'Vietnam' --> string('bc')

```



```

|--'profiles' -->
|   *'currentprofile' --> string('dhcp')
|
|--'custom' -->
|
|--'dhcp' -->
|   |
|   |--'interfaces' -->
|   |   |
|   |   |--'ethernet' -->
|   |   |   *'device' --> string('/dev/net/tulip/0')
|   |   |   *'dhcp' --> string('1')
|   |   |
|   |   |--'loopback' -->
|   |   |   *'address' --> string('127.0.0.1')
|   |   |   *'device' --> string('loop0')
|   |   |   *'dhcp' --> string('0')
|   |   |   *'flags' --> string('loopback fastloop noarp')
|   |   |   *'mask' --> string('255.0.0.0')
|   |   |
|   |   |--'ppp' -->
|   |   |   *'device' --> string('/dev/ports/serial2')
|   |   |   *'dhcp' --> string('0')
|   |   |   *'flags' --> string('ptp')
|   |   |
|   |   |--'ppp' -->
|   |   |   |
|   |   |   |--'interfaces' -->
|   |   |   |   |
|   |   |   |   |--'ethernet' -->
|   |   |   |   |   *'device' --> string('/dev/net/tulip/0')
|   |   |   |   |   *'dhcp' --> string('0')
|   |   |   |   |
|   |   |   |   |--'loopback' -->
|   |   |   |   |   *'address' --> string('127.0.0.1')
|   |   |   |   |   *'device' --> string('loop0')
|   |   |   |   |   *'dhcp' --> string('0')
|   |   |   |   |   *'flags' --> string('loopback fastloop noarp')
|   |   |   |   |   *'mask' --> string('255.0.0.0')
|   |   |   |   |
|   |   |   |   |--'ppp' -->
|   |   |   |   |   *'device' --> string('/dev/ports/serial2')
|   |   |   |   |   *'dhcp' --> string('0')
|   |   |   |   |   *'dial_prefix' --> string('')
|   |   |   |   |   *'flags' --> string('ptp')
|   |   |   |   |   *'hidden_init' --> string('')
|   |   |   |   |   *'modem_init' --> string('AT')
|   |   |   |   |   *'password' --> string('baron')
|   |   |   |   |   *'phone_number' --> string('16503252485')
|   |   |   |   |   *'ppptype' --> string('serial')
|   |   |   |   |   *'user_name' --> string('bigmpl')
|   |   |   |
|   |   |   |--'ppp800' -->
|   |   |   |   |
|   |   |   |   |--'interfaces' -->
|   |   |   |   |   |
|   |   |   |   |   |--'ethernet' -->
|   |   |   |   |   |   *'device' --> string('/dev/net/tulip/0')

```

```

|         *'dhcp' --> string('0')
|
+-'loopback' -->
|         *'address' --> string('127.0.0.1')
|         *'device' --> string('loop0')
|         *'dhcp' --> string('0')
|         *'flags' --> string('loopback fastloop noarp')
|         *'mask' --> string('255.0.0.0')
|
+-'ppp' -->
|         *'device' --> string('/dev/ports/serial2')
|         *'dhcp' --> string('0')
|         *'dial_prefix' --> string('9,')
|         *'flags' --> string('ptp')
|         *'hidden_init' --> string('')
|         *'modem_init' --> string('AT')
|         *'password' --> string('bigmpl')
|         *'phone_number' --> string('18005551212')
|         *'ppptype' --> string('serial')
|         *'user_name' --> string('baron')
|
+-'pppoe' -->
|
+-'interfaces' -->
|
+-'ethernet' -->
|         *'device' --> string('/dev/net/tulip/0')
|         *'dhcp' --> string('0')
|
+-'loopback' -->
|         *'address' --> string('127.0.0.1')
|         *'device' --> string('loop0')
|         *'dhcp' --> string('0')
|         *'flags' --> string('loopback fastloop noarp')
|         *'mask' --> string('255.0.0.0')
|
+-'ppp' -->
|         *'device' --> string('/dev/net/tulip/0')
|         *'dhcp' --> string('0')
|         *'flags' --> string('ptp')
|         *'password' --> string('bigmpl')
|         *'ppptype' --> string('pppoe')
|         *'user_name' --> string('baron')
|
+-'staticip' -->
|         *'dns_domain' --> string('be.com')
|         *'primary_dns' --> string('207.155.183.72')
|         *'secondary_dns' --> string('206.173.119.72')
|
+-'interfaces' -->
|
+-'ethernet' -->
|         *'address' --> string('192.168.1.69')
|         *'device' --> string('/dev/net/tulip/0')
|         *'dhcp' --> string('0')
|         *'mask' --> string('255.255.255.0')
|
+-'loopback' -->
|         *'address' --> string('127.0.0.1')

```

```

|         *'device' --> string('loop0')
|         *'dhcp' --> string('0')
|         *'flags' --> string('loopback fastloop noarp')
|         *'mask' --> string('255.0.0.0')
|
|--'ppp' -->
|         *'device' --> string('/dev/ports/serial2')
|         *'dhcp' --> string('0')
|         *'flags' --> string('ptp')
|
+--'route' -->
|         *'address' --> string('192.168.1.1')
|         *'interface' --> string('/dev/net/tulip/0')
|
+--'control' -->
|
+--'DNS' -->
|         *'domain' --> string('')
|         *'primary' --> string('207.155.183.72')
|         *'secondary' --> string('206.173.119.72')
|
+--'status' -->
|         *'hostname' --> string('gpz.be.com')
|         *'profile' --> string('dhcp')
|
+--'interfaces' -->
|
|         +--'/dev/net/tulip/0' -->
|         |         *'address' --> string('192.168.1.100')
|         |         *'bandwidth' --> number(10000000.000000)
|         |         *'hwaddr' --> string('00:c0:f0:37:63:ec')
|         |         *'index' --> number(1.000000)
|         |         *'mask' --> string('255.255.255.0')
|         |         *'status' --> string('up')
|         |         *'type' --> string('ethernet')
|         |
|         +--'loop0' -->
|         |         *'address' --> string('127.0.0.1')
|         |         *'bandwidth' --> number(10000000.000000)
|         |         *'hwaddr' --> string('zen-like')
|         |         *'index' --> number(2.000000)
|         |         *'mask' --> string('255.0.0.0')
|         |         *'status' --> string('up')
|         |         *'type' --> string('loopback')
|         |
|         +--'ppp0' -->
|         |         *'address' --> string('10.1.1.1')
|         |         *'bandwidth' --> number(56000.000000)
|         |         *'hwaddr' --> string('unknown')
|         |         *'index' --> number(4.000000)
|         |         *'linkmsg' --> string('ok')
|         |         *'linkstatus' --> string('disconnected')
|         |         *'mask' --> string('255.0.0.0')
|         |         *'status' --> string('down')
|         |         *'type' --> string('ppp')
|         |
|         +--'route' -->
|         |         *'address' --> string('192.168.1.1')
|         |         *'interface' --> string('/dev/net/tulip/0')

```

```

'service/network/profiles' --> ''
|
| *'currentprofile' --> string('dhcp')
|
+--'custom' -->
|
+--'dhcp' -->
|
| +--'interfaces' -->
| |
| | +--'ethernet' -->
| | |
| | | *'device' --> string('/dev/net/tulip/0')
| | | *'dhcp' --> string('1')
| | |
| | +--'loopback' -->
| | |
| | | *'address' --> string('127.0.0.1')
| | | *'device' --> string('loop0')
| | | *'dhcp' --> string('0')
| | | *'flags' --> string('loopback fastloop noarp')
| | | *'mask' --> string('255.0.0.0')
| | |
| | +--'ppp' -->
| | |
| | | *'device' --> string('/dev/ports/serial2')
| | | *'dhcp' --> string('0')
| | | *'flags' --> string('ptp')
| | |
| | +--'ppp' -->
| | |
| | | +--'interfaces' -->
| | | |
| | | | +--'ethernet' -->
| | | | |
| | | | | *'device' --> string('/dev/net/tulip/0')
| | | | | *'dhcp' --> string('0')
| | | | |
| | | | +--'loopback' -->
| | | | |
| | | | | *'address' --> string('127.0.0.1')
| | | | | *'device' --> string('loop0')
| | | | | *'dhcp' --> string('0')
| | | | | *'flags' --> string('loopback fastloop noarp')
| | | | | *'mask' --> string('255.0.0.0')
| | | | |
| | | | +--'ppp' -->
| | | | |
| | | | | *'device' --> string('/dev/ports/serial2')
| | | | | *'dhcp' --> string('0')
| | | | | *'dial_prefix' --> string('')
| | | | | *'flags' --> string('ptp')
| | | | | *'hidden_init' --> string('')
| | | | | *'modem_init' --> string('AT')
| | | | | *'password' --> string('bigmpl')
| | | | | *'phone_number' --> string('16503252485')
| | | | | *'ppptype' --> string('serial')
| | | | | *'user_name' --> string('baron')
| | | | |
| | | +--'ppp800' -->
| | | |
| | | | +--'interfaces' -->
| | | | |
| | | | | +--'ethernet' -->
| | | | |

```

```

|         *'device' --> string('/dev/net/tulip/0')
|         *'dhcp' --> string('0')
|
|--'loopback' -->
|         *'address' --> string('127.0.0.1')
|         *'device' --> string('loop0')
|         *'dhcp' --> string('0')
|         *'flags' --> string('loopback fastloop noarp')
|         *'mask' --> string('255.0.0.0')
|
|--'ppp' -->
|         *'device' --> string('/dev/ports/serial2')
|         *'dhcp' --> string('0')
|         *'dial_prefix' --> string('9,')
|         *'flags' --> string('ptp')
|         *'hidden_init' --> string('')
|         *'modem_init' --> string('AT')
|         *'password' --> string('bigmpl')
|         *'phone_number' --> string('18005551212')
|         *'ppptype' --> string('serial')
|         *'user_name' --> string('baron')
|
|--'pppoe' -->
|
|--'interfaces' -->
|
|--'ethernet' -->
|         *'device' --> string('/dev/net/tulip/0')
|         *'dhcp' --> string('0')
|
|--'loopback' -->
|         *'address' --> string('127.0.0.1')
|         *'device' --> string('loop0')
|         *'dhcp' --> string('0')
|         *'flags' --> string('loopback fastloop noarp')
|         *'mask' --> string('255.0.0.0')
|
|--'ppp' -->
|         *'device' --> string('/dev/net/tulip/0')
|         *'dhcp' --> string('0')
|         *'flags' --> string('ptp')
|         *'password' --> string('bigmpl')
|         *'ppptype' --> string('pppoe')
|         *'user_name' --> string('baron')
|
|--'staticip' -->
|         *'dns_domain' --> string('be.com')
|         *'primary_dns' --> string('10.0.0.2')
|         *'secondary_dns' --> string('10.0.0.4')
|
|--'interfaces' -->
|
|--'ethernet' -->
|         *'address' --> string('192.168.1.69')
|         *'device' --> string('/dev/net/tulip/0')
|         *'dhcp' --> string('0')
|         *'mask' --> string('255.255.255.0')
|
|--'loopback' -->

```

```
| |      *'address' --> string('127.0.0.1')
| |      *'device' --> string('loop0')
| |      *'dhcp' --> string('0')
| |      *'flags' --> string('loopback fastloop noarp')
| |      *'mask' --> string('255.0.0.0')
| |
| +- 'ppp' -->
| |      *'device' --> string('/dev/ports/serial2')
| |      *'dhcp' --> string('0')
| |      *'flags' --> string('ptp')
|
+- 'route' -->
    *'address' --> string('192.168.1.1')
    *'interface' --> string('/dev/net/tulip/0')
```

The Binder

The binder Program

API Type: Command line program

Description: Communicates with the Binder

Declared in: /boot/beos/bin/binder

binder is a command-line program that communicates with the Binder. **binder** lets you examine the structure of the Binder tree, and lets you get, set, and create properties (permissions allowing). Currently, it doesn't let you create new nodes or let you examine a property's permissions.

binder is provided as a debugging tool. Its primary usefulness is in letting you look at the branches and values in the Binder tree, particularly if you're running **Wagner** at the same time. For example, you can invoke a Binder-setting part of the **Wagner** interface, and then use **binder** to check that the setting value was properly implanted. Using **binder** to set or create properties, except as a learning experience, is a misuse of the tool.

1 Running binder

To run **binder**, the **smooved** daemon must be running and you must have a **bterm** or **Terminal** window open. Note that Wagner doesn't have to be running to use **binder**.

Note: If you're on a BeIA device, type **alt+q** to drop into the desktop environment, and then bring up a **bterm** window. The device must be in **tools** mode for this to work. See **Boot Mode** for details.

Type **binder** in the **bterm**/**Terminal** window to execute a Binder request. **binder** doesn't set up an "environment"—it executes the request and immediately returns you to the terminal's prompt.

2 Syntax

```
binder [ dir | rdir | get | put ] property [ value ]
```

binder executes a value setting or getting operation on the specified Binder *property*. The operations are:

- **dir** - list the properties in *property* (which must be a node). If you don't specify a *property*, the subroot nodes are listed.
- **rdir** - display a tree of nodes and properties starting at *property* (which must be a node).
- **get** - print the value of *property* (which can be a simple property or a node).
- **put** - set the value of *property* (which must be a simple property) to *value*.

property is given as a slash-delimited path, with the root node (`beos.binder`) assumed. For example, to list the properties in the `beos.binder.service.web`, you would type this:

```
$ binder dir service/web
```

You can't use index notation (i.e. no `node[n]` expressions), but you can represent the current user account as `~`:

```
$ binder dir user/~
```

To protect a property name that contains whitespace, protect the name with single quotes (not double quotes!):

```
$ binder get user/~bookmarks/'Welcome to Be!'
```

value is taken as a string. Use quotes (single or double) to protect the string if it contains whitespace.

2.1 Listing Properties

The `dir` operator lists the properties in a node. Each listed property is given as

```
'name' --> 'value'
```

Properties that have a value of `"<object>"` are nodes. Properties that are "unknown!" were declared as undefined in the XML root file. For simple properties, the actual value is given (in single quotes).

For example, the listing for `service.web` looks like this:

```
$ binder dir service/web
'cache' --> '<object>'
'macros' --> '<object>'
'navigator' --> '<object>'
'proxy' --> '<object>'
'security' --> '<object>'
'state' --> unknown!
```

The `state` property was declared as undefined; all other properties are nodes.

The listing for `service.system` (which contains simple properties) will look something like this:

```
$ binder dir service/system
'altq' --> '1'
'bootmode' --> 'tools'
'build' --> '2000.09.11'
'product' --> 'BeIA'
```

If you exclude the property argument, you get the listing for the root node:

```
$ binder dir
'application' --> '<object>'
'service' --> '<object>'
'user' --> '<object>'
```

2.2 Displaying a Tree

The `rdir` operator displays a tree of nodes and properties in a node. For example, this command...

```
$ binder rdir service/web/security/web
```

...produces this output:

```
'service/web/security' --> '<object>'
|
+-'acl' --> '<object>'
```



```
| |
| +- 'custom_content' --> '<object>'
| |   *'custom_content' --> 'grant'
| |   *'mail_workers' --> 'grant'
| |
| +- 'intranet' --> '<object>'
| |   *'custom_content' --> 'grant'
| |
| +- 'internet' --> '<object>'
| |   *'all' --> 'grant'
|
and so on
```

2.3 Getting a Property Value

To get the value of a single property, use the `get` operator:

```
$ binder get service/system/bootmode
tools
```

The value is given unquoted. Nodes are given as “<object>”:

```
$ binder get service
<object>
```

2.4 Setting a Property Value

To set the value of a single property, use the `put` operator:

```
$ binder put user/~/.bookmarks/'Search Engine' http://www.google.com
```

If the property doesn’t exist and the parent node is writable, the property is created.

3 Updating the Binder XML File

Just like any other Binder data, properties added or modified by **binder** are written to the Binder’s backing store, according to the rules given in “**2.3** File Synchronization” in *Introduction to the Binder*.

