< prev contents next >

3 Database Access

Chapter Index

The Secure Controls Framework's naming service provides access to three kinds of data stored in the database:

- ACNET Devices;
- ACNET Parameter Pages;
- Universal Repository of Java Objects.

3.1 Using JNDI

Secure Controls Framework uses Java Naming & Directory Interface (JNDI) as the client-side API for database access. Actually, there is a complex internal implementation behind this interface, that provides necessary links between server and clients, but the user do not have to think about it. For each data repository, there is a pluggable module on the server side, that knows how to translates user's requests from JNDI to SQL queries and feeds the data back.

JNDI defines a common mechanism for the naming service access. In our case, the directories (branches of a tree) are represented by `DirContext`. Terminal items can be instances of any serializable class. The root directory is either `InitialContext` or `InitialDirContext`. In order to specify which factory should be used to obtain the particular tree information, `Context.INITIAL_CONTEXT_FACTORY` property needs to be set up. `DirContext` and `Context` classes have a bunch of methods to browse the tree. Here is an example that gets an ACNET device description from the database:

```
// Setting up default naming factory...
System.setProperty( Context.INITIAL_CONTEXT_FACTORY,
  "gov.fnal.controls.scf.naming.InitialContextFactory" );

Context ctx = new InitialContext();                    // Creating root context
Device dev = (Device)ctx.lookup( "device/m:outtmp" );  // Getting an item
```

`Device` class will be described in §4.5. The more comprehensive example is `NamingTest` class:

[Start SCF Naming Test]
you need to have the cached Kerberos ticket!

SCF uses the following naming convention:

- Names are case-insensitive and parsed from the left to the right;
- Name elements are separated by the forward slash (`/`); as soon as all JNDI names are relative to a current context, the leading slash has no sense and is always trimmed out;
- Any characters can be used in the names, except asterisks (`*`) and percents (`%`); the semicolon (`;`) is allowed but internally substituted by the colon (`:`); spaces are allowed inside individual name elements,

leading and trailing spaces are trimmed;

Note, that the JNDI context implementation caches all the items gathered from the server. In order to refresh an item, the parent context of this item (or any predecessor) must be closed first with `close()` method.

3.2 ACNET Devices

Root context name is *device*. All devices are located on one level immediately after the root context. Thus, the full name to a device is going to be `device/<device-name>`, where `device-name` is an unqualified name of the device, with the colon on the second position and without an array index; for example: *device/m:outtmp*. An alternative way to address a device is to use its device index (DI): `device/#<di>`. Note, however, that despite the *i:beam*'s index is 178212, *device/i:beam* and *device/#178212* are two separate identical device in the context.

The `lookup()` method returns an `AtomicDevice` instance for the regular device; or a `CompositeDevice` for the family device, that contains several `AtomicDevice`s inside. In both cases, the returned atomic devices have:

- Extended device description in `DeviceAttributes` class: device index, description, node name, out-of-service flag, and sibling names;
- Full set of available properties (*reading*, *setting*, etc.);
- Extended property description in `DevicePropertyAttributes` for each property: array size, FTD, and the default event; for *control* property—also the list of defined operations.

Device usage is described in §4.5.

The device information is read-only and available for all authenticated users. Deleted and obsolete devices are not shown. The items do not have JNDI attributes. Supported methods on a context are `lookup()` and `search()` (by name). In a search filter, the asterisk can be used as a wildcard, for example: `name=m:out*`. The `search` operation can return no more than 200 resilts, otherwise generating `SizeLimitExceededException`. Use caution: as on December 2004, there are 128,840 valid devices in the database.

3.3 Parameter Pages

Root context name is either *page* or *acnet* (these notations are identical). Under the root, there are around 160 one-, two-, and three-level subtrees. For example, *page/C94* has one inner level, *page/W97* has three; most pages have two inner levels. Terminal items in these trees are `CompositeDevice`s. Each composite device represents a single parameter page: inside, it has an ordered list of `AtomicDevice`s and `EmptyDevice`s (text comments). The path to a parameter page depends on its particular location inside the inner tree, for example: *page/c94/1*, *page/l3/grads/1*, *page/w97/pag1/cryo/1*.

The returned atomic devices satisfy all the requirements from the §3.2, with two exceptions:

1. Only *analog alarm*, *control*, *reading*, *setting*, and *status* properties are provided;
2. They can have nonzero array indices;

Parameter pages are read-only and available for all authenticated users. Empty pages are not shown. The items have common attributes. Supported methods on a context are `lookup()`, `list()`, `listBindings()`, and `getAttributes()`.

3.4 Universal Repository of Objects

Universal repository of objects was designed to store any instances of serializable Java classes. Root context names can be random; currently registered are *object* and *java*. Caution should be used when updating public API of stored classes, in order to avoid serialization exceptions.

The repository supports all methods of `DirContext` interface. Each item in the repository (both contexts and serialized objects) has a set of JNDI attributes that can be obtained with `getAttributes()` method. The attributes can be updated by the owner of this item using `modifyAttributes()` methods.

Common Attributes

Attribute Name	# Values	Value Class	Meaning
`owner`	1	`java.lang.String`	Owner's user name.
`group`		`java.lang.String`	Group name.
`permissions`		`java.lang.Integer`	Access permissions in Unix format [rwxrwxrwx].
`date-modified`		`java.lang.Date`	Date of the latest update.
`owned`		`java.lang.Boolean`	*`TRUE`* if the current user is the owner of this item; *`FALSE`* otherwise.
`description`		`java.lang.String`	Optional text description.

The repository uses a UNIX-like authorization algorithm. Users, groups, and the group membership are defined in the Application Index database.

Permissions System

Operation	Required Permissions
`lookup`	`R` for the requested item, `X` for all predecessors.
`getAttributes`	
`bind`	`WX` for the parent, `X` for other predecessors.
`rebind`	`W` for the item, `X` for all predecessors.
`rebind`	`WX` for the new and old parents, `X` for all other predecessors.
`search`	`RX` for the current context, `X` for all predecessors.
`list`	
`listBindings`	
`modifyAttributes`	Must be the owner.

The **Data Browser** application uses another ad hoc data provider. Its root context *survey* has two subcontexts: *acnet* and *java*. They are mapped to the parameter page database and the Universal Repository, correspondingly, and work as described above.