Cardinal-Anticheat Dialects

Dialects

A dialect is used to abstract the database handling by loading the dialect in form of an addon. Creating your own dialect secures the data usage and guarantees the protection of sensitive data, since Cardinal has neither access to the connection nor to any credentials which are typically required. In addition, this gives you the opportunity to migrate to any form of database you desire.

Creating a dialect

Creating a dialect requires the implementation of `PunishDialect` and/or `ViolationDialect`. Both of the interfaces are generating required methods for their usage.

PunishDialect

The `PunishDialect` is used for banning and accessing ban data

```
import me.clientastisch.extension.impl.dialects.PunishDialect;

public class Punishment implements PunishDialect {

    /**
     * Punish a player
     *
     * @param uniqueId UUID of the player
     * @param reason reason of the punishment
     * @param replay replay id
     * @param minutes the punish duration
```

```
     */
    @Override
    public void punish(String uniqueId, String reason, @Nullable String replay,
long minutes) {

    }

    /**
     * Unban a player
     *
     * @param uniqueId UUID of the player
     */
    @Override
    public void pardon(String uniqueId) {

    }

    /**
     * Check if the player is banned
     *
     * @param uniqueId UUID of the player
     * @return whether the player is banned or not
     */
    @Override
    public boolean isBanned(String uniqueId) {

    }

    /**
     * Returns the Unix time when the ban expires
     *
     * @param uniqueId UUID of the player
     * @return the Unix time when the ban expires
     */
    @Override
    public long getExpire(String uniqueId) {

    }

    /**
     * Returns the replay id
     *
     * @param uniqueId UUID of the player
     * @return the replay id
     */
    @Override
    public String getReplay(String uniqueId) {
```

```
    }

    /**
     * Returns the reason for a punishment
     *
     * @param uniqueId UUID of the player
     * @return the reason
     */
    @Override
    public String getReason(String uniqueId) {

    }
}
```

ViolationDialect

The `ViolationDialect` is used to store all flags a player has thrown

```
import me.clientastisch.extension.impl.dialects.ViolationDialect;

public class Violation implements ViolationDialect {

    /**
     * Add numerus violations to the players
     * violation history
     *
     * @param uniqueId UUID of the player
     * @param violations List of violations
     */
    @Override
    public void addViolations(String uniqueId, List<String> list) {

    }

    /**
     * Remove violations which have been collected so far
     *
     * @param uniqueId UUID of the player
     */
    @Override
    public void removeViolations(String uniqueId) {

    }

    /**
```

```
     * Returns a list of collected violations
     *
     * @param uniqueId UUID of the player
     * @return List of violations
     */
    @Override
    public List<String> getViolations(String uniqueId) {

    }

    /**
     * Returns all uniqueIds which have
     * at least one violation stored
     *
     * @return List of uniqueId
     */
    @Override
    public List<String> getViolations() {

    }
}
```

Register a dialect

Registering a dialect is similar to any other event or command you're trying to register. Therefore you need to get the static reference of `Extension` and call the method `registerDialect` with the required arguments.

```
import me.clientastisch.extension.Extension;
import me.clientastisch.extension.impl.Addon;

public class Core implements Addon {

    @Override
    public void onEnable() throws Exception {
        Extension.registerDialect(this, new Punishment());
        Extension.registerDialect(this, new Violation());
    }

    @Override
    public void onDisable() throws Exception {

    }
}
```