MissionPouches

Commands & Permissions

MissionPouches has a variety of commands and permissions, allowing you to give pouches to your players, list available expansions, reload the plugin and more.

Note: <> are mandatory arguments, and [] are optional arguments.

- /missionpouches: Root command. Will print help page.
  - give <player> <pouch> [amount] [finished]: Give a pouch to a player. Finished is a boolean, set it as "true" if you'd like to pouch to be completed.
  - giverandom <player> [finished]: Give a random pouch to a player. Pouches with weight 0 won't be considered.
  - giveall <pouch> [amount] [finished]: Give all online players a pouch.
  - expansions: List all installed expansions.
  - list: List all loaded pouches.
  - reload: Reload the plugin.

Permissions

- missionpouches.admin.give - Access to /missionpouches give.
- missionpouches.admin.giverandom - Access to /missionpouches giverandom.
- missionpouches.admin.giveall - Access to /missionpouches giveall.
- missionpouches.admin.expansions - Access to /missionpouches expansions.
- missionpouches.admin.list - Access to /missionpouches list.
- missionpouches.admin.reload - Access to /missionpouches reload.

NOTE: Every pouch can have its own permission, and random actions can have permission checks as well.

Creating a Pouch

Creating a pouch is relatively simple once you understand the configuration.

This is a full example pouch configuration file:

```
name: Example
weight: 0.1

lockedItem:
  material: DIAMOND_BLOCK
  name: '<red><bold>Example Pouch'
  lore:
    - '<gray>Break 100 cobblestone blocks and'
    - '<gray>10 diamond ores to unlock this pouch!'
    - ''
    - '<yellow>Cobblestone: %progress_breakCobblestone%/%goal_breakCobblestone%'
    - '<yellow>Diamond Ores: %progress_breakDiamonds%/%goal_breakDiamonds%'
unlockedItem:
  material: DIAMOND_BLOCK
  name: '<green><bold>Example Pouch'
  lore:
    - '<gray>Break 100 cobblestone blocks and'
    - '<gray>10 diamond ores to unlock this pouch!'
    - ''
    - '<yellow>Click to open!'

singleMission: false
missions:
  breakCobblestone:
    type: BLOCK_BREAK
    goal: 100
    parameters:
      blocks:
        - COBBLESTONE
  breakDiamonds:
    type: BLOCK_BREAK
```

```
    goal: 10
    parameters:
      blocks:
        - DIAMOND_ORE

expireTime: -1

pouchUpgrade: Example2

recipe:
  shape:
    - AAA
    - ABA
    - AAA
  ingredients:
    A: GLASS
    B: NETHER_STAR

actions:
  guaranteed:
    - '[Message] text=<gray>Congratulations, you redeemed the Example Pouch!'
  randomAmount: 1
  random:
    '0':
      actions:
        - '[Message] text=<green>This green message can appear 50% of the time.'
      weight: 0.5
      requirePermission: false
      invertPermissionCheck: false
      permission: actions.check
    '1':
      actions:
        - '[Message] text=<red>This red message can appear 50% of the time.'
      weight: 0.5
      requirePermission: true
      invertPermissionCheck: false
      permission: actions.check
```

- **name:** This parameter defines the internal name of your pouch. This must be unique across all pouches.

- **weight:** The weight of your pouch when giverandom is used.
- **lockedItem & unlockedItem:** These are the items used when the pouch is locked and unlocked, respectively
  - Both of these support custom model data and skull textures. Example:
  - ```
    unlockedItem:
      material: PLAYER_HEAD
      name: '<green><bold>Example Pouch'
      headTexture: 'example'
      lore:
        - '<gray>Break 100 cobblestone blocks and'
        - '<gray>10 diamond ores to unlock this pouch!'
        - ''
        - '<yellow>Click to open!'
    ```
  - Model data example:
  - ```
    unlockedItem:
      material: PAPER
      modelData: 10000
      name: '<green><bold>Example Pouch'
      lore:
        - '<gray>Break 100 cobblestone blocks and'
        - '<gray>10 diamond ores to unlock this pouch!'
        - ''
        - '<yellow>Click to open!'
    ```
- **singleMission:** This defines whether a single mission is required to unlock the pouch, or all missions are required.
- **missions:**
  - **type:** Mission type.
  - **goal:** The goal needed to complete the mission.
  - **parameters:** Additional parameters. These vary per mission type - check the expansion information.
- **expireTime:** The amount of time before the pouch expires. This time starts to run once the pouch is given. Set to -1 to disable.
- **pouchUpgrade:** This defines the pouch that can be obtained by combining multiple of the same pouch in a crafting table. Optional feature.
- **recipe:**
  - **shape:** Shape of the recipe. All letters need to be specified in the ingredients, and needs to be in a 3x3 grid.
  - **ingredients:** Ingredients to use in the shape.
- **actions:**

- **guaranteed:** These actions will always fire. Check the Actions page for a full list of actions.
- **randomAmount:** Amount of random actions to fire.
- **random:**
  - **actions:** Actions to fire for this random object.
  - **weight:** Weight of this random object.
  - **requirePermission:** Whether this random object requires a permission to fire.
  - **invertPermissionCheck:** Whether the permission check should be inverted (so it only fires if you do not have the permission). Needs "requirePermission" to be set to true.
  - **permission:** Permission required. Needs "requirePermission" to be set to true.

If you need more help creating your pouches, feel free to reach out on Discord!

Built-in Actions

Actions allow you to define a variety of things to run whenever a pouch gets redeemed.

Actions support PlaceholderAPI and has 2 built-in placeholders:

- %player% - Replaced with the player name.
- {random:<min>-<max>} - Replaced with a random number between those bounds.

1. **Broadcast**
   - Arguments:
     - *text:* Text to send. Formatted in MiniMessage.
   - Example: "[Broadcast] text=<green>Test Broadcast"
2. **Console**
   - Arguments:
     - *cmd:* Command to execute as console.
   - Example: "[Console] cmd=say Hello!"
3. **GiveEffect**
   - Arguments:
     - *type:* Potion effect type. A list is available here: https://hub.spigotmc.org/javadocs/bukkit/org/bukkit/potion/PotionEffectType.html
     - *strength:* Strength of the effect. Starts at 0.
     - *duration:* Duration in seconds.
   - Example: "[GiveEffect] type=SPEED;;strength=2;;duration=30"
4. **GiveFood**
   - Arguments:
     - *amount:* Amount of food points to give.
   - Example: "[GiveFood] amount=5"
5. **GiveHealth**
   - Arguments:
     - *amount:* Amount of health to give. 1 health equals to 1/2 heart.
   - Example: "[GiveHealth] amount=6"
6. **Message**
   - Arguments:
     - *text:* Text to send. Formatted in MiniMessage.
   - Example: "[Message] text=<green>Test Message"
7. **Sudo**
   - Arguments:
     - *cmd:* Command to execute as player.
   - Example: "[Sudo] cmd=say Hello!"
8. **PlaySound**

- Arguments:
  - *sound:* Sound to play. This needs to be a full Minecraft sound key. Here is a list: https://minecraft.wiki/w/Sounds.json
  - *source:* Sound source. Available values: AMBIENT, BLOCK, HOSTILE, MASTER, MUSIC, NEUTRAL, PLAYER, RECORD, VOICE, WEATHER. Defaults to AMBIENT.
  - *volume:* Sound volume. Usually goes from 0.0 to 1.0. Defaults to 1.0.
  - *pitch:* Sound pitch. Usually goes from 0.0 to 1.0. Defaults to 1.0.
- Example: "[PlaySound] sound=block.note_block.bell;;source=MUSIC;;volume=1.0;;pitch=0.5"

9. **Title**
   - Arguments:
     - *title:* Title to send. Formatted in MiniMessage.
     - *subtitle:* Optional subtitle to send. Also formatted in MiniMessage.
     - *fadeIn:* Fade in time.
     - *stay:* How long the title stays on screen.
     - *fadeOut:* Fade out time.
   - Example: "[Title] title=<green>Test;;subtitle=<yellow>Test 2;;fadeIn=50;;stay=100;;fadeOut=50"

If you need more actions, contact me on Discord!

Built-in Mission Types

MissionPouches comes with a few built-in mission types.

- **BLOCK_BREAK**
  - This mission type fires when a block is broken.
  - Parameters:
    - *blocks:* List of blocks that count towards the mission. If not specified, any block will count.
- **BLOCK_PLACE**
  - This mission type fires when a block is placed.
  - Parameters:
    - *blocks:* List of blocks that count towards the mission. If not specified, any block will count.
- **CONSUME_ITEM**
  - This mission type fires when a player consumes an item such as food.
  - Parameters:
    - *items:* Items that count towards the mission. If not specified, any item will count.
- **ITEM_CRAFT**
  - This mission type fires when a player crafts an item.
  - Parameters:
    - *items:* Items that count towards the mission. If not specified, any item will count.
- **MOB_KILLS**
  - This mission type fires when a player kills a mob.
  - Parameters:
    - *entities:* Entities that count towards the mission. If not specified, any entity will count.
- **PLAYER_FISH**
  - This mission type fires when a player catches a fish.
  - No parameters available.

If you need a custom mission type, let me know on Discord!

Developer API

Repository Information

Maven:

```
<repository>
    <id>mysticalkingdoms-public</id>
    <url>https://nexus.mysticalkingdoms.com/repository/public/</url>
</repository>
```

Gradle:

```
    maven {
        url = uri('https://nexus.mysticalkingdoms.com/repository/public/')
    }
```

To create your own expansion, you first need to have a main class that will handle enabling, disabling and reloading the expansion, just like a plugin. This is an example of a main class for an expansion:

```
package com.mysticalkingdoms.pouchexpansion.cratereloaded;

import com.mysticalkingdoms.missionpouches.MissionPouches;
import com.mysticalkingdoms.missionpouches.missions.MissionExpansion;
import com.mysticalkingdoms.missionpouches.missions.MissionExpansionDescription;
import com.mysticalkingdoms.pouchexpansion.cratereloaded.types.CrateReloadedOpen;

import java.io.File;

public class CrateReloadedExpansion extends MissionExpansion {

    public CrateReloadedExpansion(MissionPouches plugin, MissionExpansionDescription expansionDescription,
File dataFolder) {
        super(plugin, expansionDescription, dataFolder);
    }

    @Override
```

```
    protected boolean onEnable() {
        registerMissionTypes(new CrateReloadedOpen(getPlugin()));
        return true;
    }

    @Override
    protected void onDisable() {}

    @Override
    protected void onReload() {}
}
```

Keep the constructor as is, otherwise your expansion will not load.

- **onEnable:** You should register any logic here, and then register your mission types.
- **onDisable:** Disable any logic that needs to be disabled (example: databases)
- **onReload:** This is called when /mp reload is used.

Now, for your mission type, you need to extend the class "MissionType".

There are two ways to create this class. The first one uses a Function that gives you a ConfigurationSection (the pouch configuration), and expects MissionProgress as the return value.

You must specify a mission type key. This should be unique. A good idea is to prefix it with the name of your plugin/expansion.

```
package com.mysticalkingdoms.pouchexpansion.cratereloaded.types;

import com.hazebyte.crate.api.event.CrateRewardEvent;
import com.mysticalkingdoms.missionpouches.MissionPouches;
import com.mysticalkingdoms.missionpouches.missions.MissionProgress;
import com.mysticalkingdoms.missionpouches.missions.MissionType;
import org.bukkit.event.EventHandler;

import java.util.List;

public class CrateReloadedOpen extends MissionType {
    public CrateReloadedOpen(MissionPouches plugin) {
        super(plugin, "CRATERELOADED_OPEN");
    }

    @EventHandler
```

```
    public void onReward(CrateRewardEvent event) {
        checkPouches(event.getPlayer(), section -> {
            if (section == null) {
                return MissionProgress.of(1L, true);
            }

            List<String> crates = section.getStringList("crates");
            if (crates.isEmpty()) {
                return MissionProgress.of(1L, true);
            }

            return MissionProgress.of(crates.contains(event.getCrate().getCrateName()) ? 1L : 0L, true);
        });
    }
}
```

MissionType implements Listener, so you can listen to any Bukkit events. In the example above, we're listening to CrateRewardEvent, and then calling checkPouches on the player, we get the ConfigurationSection for the pouch configuration, we do our checks, and then finally return MissionProgress. MissionProgress has 2 static methods:

- *of(long progress, boolean add):* This method sets the progress to the specified value. If the add boolean is set to true, then it will add progress instead of setting.
- *noProgress():* This method will just return no progress.

Notes:

1. The ConfiguratIonSection from the function can be null, for example, if the user didn't specify any parameters.

Now, there's a second version which gives you access to the NBTItem of the pouch when it's being checked.

This is an example of an expansion that uses it:

```
package com.mysticalkingdoms.pouchexpansion.uniquekills.types;

import de.tr7zw.changeme.nbtapi.NBTList;
import com.mysticalkingdoms.missionpouches.MissionPouches;
import com.mysticalkingdoms.missionpouches.missions.MissionProgress;
import com.mysticalkingdoms.missionpouches.missions.MissionType;
import org.bukkit.entity.Player;
import org.bukkit.event.EventHandler;
```

```
import org.bukkit.event.entity.PlayerDeathEvent;

import java.util.UUID;

public class UniqueKillsType extends MissionType {
    public UniqueKillsType(MissionPouches plugin) {
        super(plugin, "UNIQUEKILLS_KILL");
    }

    @EventHandler
    public void onKill(PlayerDeathEvent event) {
        Player killer = event.getEntity().getKiller();
        if (killer == null) {
            return;
        }

        checkPouches(killer, (section, nbtItem) -> {
            NBTList<UUID> list = nbtItem.getUUIDList("PlayersKilled");
            if (list.contains(event.getEntity().getUniqueId())) {
                return MissionProgress.noProgress();
            }

            list.add(event.getEntity().getUniqueId());
            return MissionProgress.of(1L, true);
        });
    }
}
```

This allows you to query the item for specific things, to make your expansions even more flexible. For example, the UniqueKills expansion, available on Discord, uses this to write NBT data of which players have been killed, so that you have to kill unique players to progress through the mission.

If you have any questions, feel free to ask on Discord!