RP Content Modding

This guide contains the information you require to create and modify RP Content add-ons for Blockland. All functions and information that do not exist in a default Blockland installation will be listed below. Some functions and commands will not be listed below due you probably will not need them. But if you do, then read through the add-on script for that function or command.

Table of content

RP Core

This mod is the core of the RP Content and contains saving, experience, GUI functions and most system functions that is required for an RP to work. This is also the start-node for all other extensions made to it.

Brick

PlantBrick(brick)

This is called right after a brick is planted. Like `onPlant`, but scheduled right after to avoid brick ownership.

LoadingBrick(brick)

Like `PlantBrick`, but when you are loading a save this will be called instead.

RemoveBrick(brick)

Like `PlantBrick`, but when removing it, but is called instantly and on `onDeath`.

Game

BootRP()

This starts the actual RP. Normally, every mod will have this in a package with `Parent::BootRP();` and a check if the RP is already activated.

```
function BootRP()
{
   if ($RP::activated)
      return;
   Parent::BootRP();

   // The rest of your starting code goes here
}
```

ShutRP()

This stops the actual RP. Normally, every mod will have this in a package with `Parent::ShutRP();` and a check if the RP is not activated.

```
function ShutRP()
{
   if (!$RP::activated)
      return;
   Parent::ShutRP();

   // The rest of your cleaning up code goes here
}
```

GUI Controls

These controls are only here to make your job of adding stats without having to parent everything.

RP_AddStat(name, var, info, varList, prefix, suffix, hide)

Adds a stat to the stat system that makes it possible for the user to see it.

```
// This adds a stat with the name Money accessing the db value money
that will be displayed for the user all the time while passing the
output to the list $RP::list::money with the prefix $ and no suffix
and wont display if the output is empty
RP_AddStat("Money", "money", 1, "RP::list::money", "$", "", 1);
```

Init load

It is recommended to preload preferences, bricks and items when the server starts, and therefore keeps them in a separate file of your choice.

Initialize

A normal file would look like this. Note that you have to load your preferences first, and then call the function `LoadPreferences();`.

```
// Load bricks
LoadOre();
LoadGem();
LoadTree();
LoadFish();

// RP Money
if (RPModExist("Money") || $AddOn__RP_Money == 1)
   exec("./bricks/sell.cs");

// Load items
exec("./items/rod.cs");

// Standard
exec("./prefs.cs");

// Reload preferences
LoadPreferences();

// Create RTB Preferences
if (isFile("Add-Ons/System_ReturnToBlockland/server.cs") &&
!$RP::Resources::RTB)
{
   if (!$RTB::RTBR_ServerControl_Hook)
   {
      exec("Add-
Ons/System_ReturnToBlockland/RTBR_ServerControl_Hook.cs");
   }
   $RP::Resources::RTB = true;

   // Server
   RTB_registerPref("Experience Mining Ore", "RP Content",
"$RP::pref::server::hardcoreExp", "int 1 5000", "RP Resources",
$RP::pref::server::hardcoreExp, false, true, "");
   // And more prefs
}
```

Saver

The Saver can be used to store various sorts of data, but is most specialized on player data. This is due that it will know who is online or not, if used properly.

GameConnection::getSaveKey()

This is used so you may reach the correct row without having to think of which mode the user is using.

```
%client.getSaveKey();
```

To reach the RP information for each player, you should use this method.

Get from DB

```
RPDB.get(%client.getSaveKey(), "name");
```

This is the same when getting information.

Set in DB

```
RPDB.set(%client.getSaveKey(), "name", "Badspot");
```

It is also possible to reach the data directly when the player is online, by using this method.

Get from client

```
%client.RPData.value["name"];
```

RPDB.addValue(value , default_value)

Adds a `value` with a `default_value` to the database. If the default value is changed it will update.

In Saver v1.5 a new ability was added the ability to save a Boolean value that will be compressed for better storage.

RPDB.addBool(value , default_value)

Adds a bool `value` with a `default_value` to the database. If the default value is changed it will update. This is used when adding a bool to the bool list that will be passed through a compressor for better storage.

Get bool from DB

```
RPDB.getBool(%client.getSaveKey(), "storage");
```

Set bool in DB

```
RPDB.setBool(%client.getSaveKey(), "storage", 1);
```

The same goes when gathering the data from each client.

Get bool from client

```
%client.RPData.getBool("storage");
```

Set bool in client

```
%client.RPData.setBool("storage", 1);
```

Default server.cs

This can look different from person to person. Although, you always need to do two things here that have to be called.

1. Require RP Core. Also if other mods are required, you should also add them.
2. Register your mode with RPRegMod function.

Default server.cs

```
%error = ForceRequiredAddon("RP_Core");

if (%error == $Error::AddOn_Disabled)
{
   warn("RP_Add-On: RP_Core has been enabled.");
}
if (%error == $Error::AddOn_NotFound)
{
   error("RP_Add-On: RP_Core is missing and is required by this Add-On.");
}
// Final check
else if (isFunction(RPModExist) && isFunction(CreateRPDB))
{
   // Register mod
   RPRegMod("Add-On");

   // Execute files
}
```

System

RPRegPackage(package)

Register a package that will be activated when the RP starts and deactivated when the RP ends. Use if you are packaging Blockland critical systems that affects the actual gameplay of Blockland.

```
RPRegPackage("RP_Core_System");
```

RPRegMod(mod_name)

Register a mod to the RP.

```
RPRegMod("Resources");
```

RPModExist(mod_name)

Returns true if mod exists.

```
echo(RPModExist("Resources"));
```

RPModsExists(mod_names)

Returns true if all listed mods exist.

```
echo(RPModsExists("Resources Money Jobs"));
```

DisplayFile(client , file)

Reads a file and displays it to the client.

AppendToFile(file , text)

Appends `text` to a file.

GamePreloadFile(file)

Loads a file and temporarily stores the information to `$Pre::`.

File: ore.dat

```
TYPE ORE
 name Iron
 resources 1
 hardCore 50
 colorHasRes 48
 colorHasNoRes 50
 oreIsBack 30
 adminOnly 0
```

```
GamePreloadFile("./ore.dat");
// This variable contains 48
$Pre::colorHasRes;
```

GameReadFile(file)

Reads a file and stores everything to `$File::line[`*n*`]`, where *n* is the line in that file.

GameTransferFile(file_source , file_target)

Transfers `file_source` to `file_target`.

CheckFileVersion(file)

Get the version number of the file. The version system works that the first line has to be dedicated to the version system by the first character has to be v, and then the number.

File: ore.dat

```
v2
TYPE ORE
 name Iron
 resources 1
 hardCore 50
 colorHasRes 48
 colorHasNoRes 50
 oreIsBack 30
 adminOnly 0
```

```
echo(CheckFileVersion("MyFile.dat"));
```

findVowel(text)

Returns "an" if found any of these characters in the beginning of `text`: a, e, i, o, u or y

getNumberExtra(number)

```
// 23rd
Echo("23" @ getNumberExtra(23));
// 4th
Echo("4" @ getNumberExtra(4));
```

getMetricPrefix(number)

Forces the number to be below one thousand while adding a prefix according to the metric system.

```
// Returns 2.3M
echo(getMetricPrefix(2300450));
```

addToList(list , data)

Adds `data` to `list`.

checkInList(list , data)

Controls if `data` exists in `list`.

removeFromList(list , data)

Removes `data` from `list`.

HexToRGBA(hex)

Returns the correct RGBA string according to a hex value.

```
// Returns "0.5 0.5 0.5 1"
echo(HexToRGBA("808080"));
```

findClientByText(text)

Get a client according to ID or name.

Client

GameConnection::EnterRPGame(client)

This is called when the `client` is joining the RP.

GameConnection::LeaveRPGame(client)

This is called when the `client` is leaving the RP.

GameConnection::FinishRPJoining(client)

This is called when the `client` has joined the RP game.

GameConnection::FinishRPSpawning(client)

This is called when the `client` is in the RP and is spawning.

GameConnection::displayInfo(client)

Display the info to the `client`.

GameConnection::updateInfo(client)

This is called to the `client` as often as the host have setup it, and could be called to prevent it from stop calling itself.

GameConnection::displayStats(client)

This is called to the `client` normally uses the command /stats.

GameConnection::CheckPrefTRust(client , rank)

Checks if the `client` has the rank where 0 is no one and 4 is anyone in the admin system.

GameConnection::getPrefTrust(client)

Gets the trust number the `client` has as mentioned above.

GameConnection::CheckLastAction(client , object)

Checks the last action according from `$RP::pref::user::lastActionTimer`.

GameConnection::ThrowLastActive(client)

Activates the automatic-disable-active-brick when activated a brick. This is already thrown each time when pressing a clickable brick, but has to be thrown each time the user is typing something to the brick.

GameConnection::StopActivate(client)

Stop the talk to the brick. This is thrown each time something went wrong or when automatic-disable-activate-brick scheduled out.

GameConnection::isInRP(client)

Returns true if the `client` is in the RP.

GameConnection::isHost(client)

Returns true if the `client` is the host of the server.

Player

Player::RP_giveDefaultEquipment(player)

Give the equipments to the `player` that is used in the RP.

Player::addTools(client , tools)

Give the tools to the `player`.

Player::getObjectFromPOW(player , mask)

Get what the `player` is looking at and returns as an object. The optional `mask` is used if you want to specify what type of object it returns.

Clickable bricks

Clickable bricks can be used to display information or interact with the RP. This guide will look at the sell resource brick.

To make a clickable brick, you have to make three things

1. The actually datablock which will contain the datablock information and `isInfoBrick` variable.
2. The function `Activate` that will be called when click on the brick.
3. The function `parseData` which receives information from player input.

fxDTSBrickData

```
datablock fxDtsBrickData(ResourceBrickData : brick2x2RampPrintData)
{
   category = "RP Admin";
   subcategory = "Admin event bricks";

   uiName = "Resource brick";

   specialBrickType = "";

   isInfoBrick = 1; // This makes it to an info brick
   adminOnly = 1; // Only admins can plant this
};
```

Activate(datablock , client)

This will be called when clicking the brick.

```
function ResourceBrickData::Activate(%datablock, %client)
{
   %ore = %client.RPData.value["ore"];
   %wood = %client.RPData.value["wood"];
   // No resources
   if (%ore <= 0 && %wood <= 0)
   {
      %client.activateDatablock = 0;
      messageClient(%client, '', "\c3You have no resources.");
      return false;
   }
   messageClient(%client, '', "\c3Resources");
   // You have ores
   if (%ore > 0)
      messageClient(%client, '', "\c31. \c6Sell ores");
   // You have wood
   if (%wood > 0)
      messageClient(%client, '', "\c32. \c6Sell logs");
   %client.activateDatablock = %datablock; // Store the datablock so
it can be used
   %client.activateState = 0; // Set this to 0
   return true;
}
```

parseData(datablock , client , text)

This will be called when a player has clicked a brick and is sending text to the server. If it returns `false`, a message ("Invalid Command") will be sent to the client.

```
function ResourceBrickData::parseData(%datablock, %client, %text)
{
   // Remember this. Also for each state.
   %client.ThrowLastActive();

   %number = mFloor(%text);
   switch (%client.activateState)
   {
      // Start
      case 0:
         switch (%number)
         {
            // Sell ores
            case 1:
               %ore = %client.RPData.value["ore"] *
$RP::pref::gain::money::ore;
               if (%ore == 0)
                  return false;

               %client.RPData.value["ore"] = 0;
               %client.RPData.value["money"] += %ore;
               messageClient(%client, '', "\c6You sold \c3ores\c6
for \c3$" @ %ore @ "\c6.");
               %client.displayInfo(); // Not necessary
               %client.stopActivate(); // Stop
            // Sell logs
            case 2:
               %wood = %client.RPData.value["wood"] *
$RP::pref::gain::money::wood;
               if (%wood == 0)
                  return false;

               %client.RPData.value["wood"] = 0;
               %client.RPData.value["money"] += %wood;
               messageClient(%client, '', "\c6You sold \c3logs\c6
for \c3$" @ %wood @ "\c6.");
               %client.displayInfo(); // Not necessary
               %client.stopActivate(); // Stop
            // Invalid
            default:
               %client.stopActivate(); // Stop
               return false; // Invalid command
            }
         // Invalid
         default:
            %client.stopActivate(); // Stop
            return false; // Invalid command
      }
   return true;
}
```

RP Money

RP Money contains the money and shop system.

Game

CheckTaxID(key)

Uses the `key` each tick from the specific user to be used to pay tax, get interest or something that is changed over time.

RequestTransfer(client , brick , service , amount , amount)

Requests a transfer for a service to the client.
Client – The client asking for the service.
Brick – The brick that will hold the money.
Service – The service the client was offered.
Amount – The amount of money requested.

BuyItem(client , brick , datablock)

Buys the item if the brick have enough money.
Client – That requested the item.
Brick – Will have the money for the item.
Datablock – The item datablock.

System

prepareDisplayMoney(amount)

Returns a modified amount of money in a better way.

RP Time

This contains all sorts of time calculations and displays the current virtual RP time.

Game

RPTick()

Depending on the host settings, this is called each tick and will increase the time.

CheckTickID(key)

Uses the `key` each tick from the specific user to do whatever he wants that tick.

SaveIDTick(identifier)

Call it as `SaveIDTick(0);` and it will save each ID online with a second in between.

System

getClockTime(hour , english_time , minutes)

Insert `hour` and if it will be `english_time` with 12 or 24 hour clock. Then `minutes` is used to tell if you want to see how many minutes have passed or not.

```
// Returns "4:00 PM"
echo(getClockTime(16, 1, 1));
```

getMinutes()

Get the virtual RP minutes passed since last tick.

GetDayOfWeek(year , month , day)

Gets the week day depending on the current date.

```
// Returns Saturday
echo(GetDayOfWeek(2010, 4, 24));
```

IsLeapYear()

Returns if it is a leap year or not. Meaning if the year has 365 days or not.

RP Resources

RP Resources contains resources as ore, tree and fish. It also has Axe, Pickaxe and Rod items.

Brick

fxDTSBrick::ResetResource(brick)
Resets the `resource` in the brick so it will contain the default ones.

RP Jobs

The basics in a job system exist in this one. Do handle with care.

Brick

fxDTSBrick::addSpawn(brick)

Add the current brick to the spawn system where people will spawn from.

RemoveSpawn(brick)

Removes the brick from the spawn system.

Game

FindSpawn(search , client)

Get the spawn according from a search name and the client.

CreateSpawns()

Creates the spawns that will be used for the system.

FindJobIDByName(name)

Returns the job ID according to the name. Returns 0 if the job or mod for the job does not exist.

```
// This can be used if you want to get the ID anyway
%i = $RP::job::indexName[%name];
%ID = $RP::job::listID[%i];
// If %ID is empty, then it is an invalid job
```

Promotion(client)

Do some tests for promotion, and if everything went fine, prompts the promotion to the `client`.

Promote(client , answer)

If Promotion was called first, the promotion still exists and the answer is yes, then the `client` will be promoted.

RP Crime

The system contains everything that has to do about crimes, like jail, police force and criminals.

Game

CommitCrime(client , reason , demerits)

Makes the `client` commit a crime for a `reason` with a special amount of `demerits`.

CheckCriminal(key)

Checks the criminal with the `key` and deducts the user demerits.

CheckPrisioner(key)

Checks the criminal with the `key` and removes one jail tick from his records.

AddToCriminalList(key)

Adds a user with the `key` to the criminal list.

AddToJailList(key)

Adds a user with the `key` to the jailing list.

AddToBountyList(key)

Adds a user with the `key` to the bounty list.

RemoveFromCriminalList(key)

Removes a user with the `key` from the criminal list.

RemoveFromJailList(key)

Removes a user with the `key` from the jailing list.

RemoveFromBountyList(key)

Removes a user with the `key` from the bounty list.

Client

GameConnection::TakeDirtyMoney(client , target , extra)

When `target` takes money from `client` that he have stolen and transfers them. The `extra` is a boolean where it tells to remove a special payment percentage too.

GameConenction::TransferDirtyMoney(client , target , amount)

Transfer a special `amount` of dirty money from `client` to `target`.

Command Table

[i/n] is [BL_ID/name].

Command	Arguments	Mod	Description
StartRP	-	Core	Starts the RP game mode.
EndRP	-	Core	Ends the RP game mode.
JoinRP	-	Core	Joins the RP game mode.
LeaveRP	-	Core	Leaves the RP game mode.
UpdateInfo	-	Core	Forces an update of the info.
ToggleCapsFilter	-	Core	Toggles the Caps filter.
ToggleReport	-	Core	Toggles the Report command.
ToggleSuggest	-	Core	Toggles the Suggest command.
ToggleLocalChat	-	Core	Toggles the local chat.
SetLocalChatRange	[bricks]	Core	Sets the local chat and name visibility range.
Admin	[message]	Core	Sends a message to all admins. (Admin only)
Report	[message]	Core	Sends a report to the server host and logs the player name, BL_ID and the message.
Suggest	[message]	Core	Same as above, but for suggestions.
Stats	[i/n]	Core	Displays the stats about the specified player or yourself.
GrantExp	[amount] [i/n]	Core	Grants experience to specified person.
DeductExp	[amount] [i/n]	Core	Removes experience from a specified person.
ClearPlayer	[i/n]	Core	Clear a user data entirely.
ClearRecord	[i/n]	Crime	Clear a user record entirely.
Group	[message]	Jobs	Sends a message to the current user group.
SetJob	[jobID] [i/n]	Jobs	Sets a jobID for the user.
GiveMoney	[amount]	Money	Gives a special amount of money to the player you are looking at.
GrantMoney	[amount] [i/n]	Money	Grants money to specific person.
DeductMoney	[amount] [i/n]	Money	Removes money to specific person.
ClearLoan	[i/n]	Money	Clear a user loan.
Date	-	Time	Displays the current date including minutes.
ForceTick	-	Time	Forces one RP hour to go.

RP Add-Ons Abbreviations

Long name	Short name	Abbreviation
RP Content	-	RPC
RP Core	Core	RPC
RP Crime	Crime	RPCr
RP Jobs	Jobs	RPJ
RP Money	Money	RPM
RP Time	Time	RPT
RP Real Estate	Real Estate	RPRE
RP GUI	RP GUI	RP GUI

Event table

Mod	Type	Name	Parameter(s)	Description
Core	Input	onRPTrue	-	When something is true.
Core	Input	onRPFalse	-	When something is false
Jobs	Output	ifJob	[job]	Control if the user have that job.
Money	Input	onTransferSuccess	-	When transferring money.
Money	Output	transferMoneyToBrick	[service] [amount]	Transfer amount of money to brick for a service.
Money	Output	transferMoneyToClient	[trust]	Transfer money back to client depending on the trust.
Money	Output	displayCurrentMoney	-	Displays current money on brick.
Time	Output	ifRPDate	[min] [hour] [day] [month] [year]	Controls a date if it is the current one.

Event examples

Below are some examples of how you can setup an event system.

Shop system

This system should be used for setting up a shop.

Default:

```
onActivate -> Self -> transferMoneyToBrick -> [Gun] [100]
ontransferSuccess -> Self -> setItem [Gun]
```

With variables:

```
onActive -> Client -> IfVariable -> [Bread] >= [1]
onVariableTrue -> Self -> transferMoneyToBrick -> [Bread] [20]
onTransferSuccess -> Client > AddVariable -> [Bread] Subtract [1]
```

Job related events

To check a job.

```
onActivate -> Self -> ifJob [Tourist]
onRPTrue -> Client -> messageClient [Welcome to our town!]
onRPFalse -> Client -> messageClient [Welcome back!]
```

Doors:

```
onActivate -> Self -> ifJob [Police]
onRPTrue -> Self -> contentStart . . .
```

Check date

This should be used to control special dates.

True if the date is 15th:

```
onActivate -> Self -> ifRPDate [-1] [0] [15] [None] [2010]
onRPTrue -> Client -> messageClient [You won!]
onRPFalse -> Client -> messageClient [Please try again..]
```