

# Getting Started

## Adding the NuGet package to your project

▼ Filter by title

You need to pull BP.AdventureFramework into your project. The easiest way to do this is to add the NuGet package. The latest package and installation instructions are available here

(<https://github.com/benpollarduk/BP.AdventureFramework/pkgs/nuget/BP.AdventureFramework>).

**Getting Started (getting-started.html)**

**Executing a Game**

**(executing-a-game.html)**

Once the package has been installed it's time to jump in and start building your first game.

+ **Locations**

**Items (items.html)**

+ **Characters**

**Commands**

**(commands.html)**

**End Conditions (end-conditions.html)**

**Conditional Descriptions (conditional-descriptions.html)**

**Frame Builders (frame-builders.html)**

```
// create the player. this is the character the user plays as
var player = new PlayableCharacter("Dave", "A young boy on a quest to find the meaning of life.");
```

```
/// create region maker. the region maker simplifies creating in game regions. a region contains a series of rooms
```

```
var regionMaker = new RegionMaker("Mountain", "An imposing volcano just East of town.");
```

**Getting Started (getting-started.html)**

```
// add a room to the region at position x 0, y 0, z 0
```

```
[0, 0, 0] = new Room("Cavern", "A dark cavern set in to the base of the mountain.");
```

**Executing a Game**

**(executing-a-game.html)**

## + Locations

```
// create overworld maker. the overworld maker simplifies creating in game overworlds. an overworld contains a series of regions
```

**Items (items.html)**

```
var overworldMaker = new OverworldMaker("Daves World", "An ancient kingdom.", regionMaker);
```

## + Characters

**Commands** the callback for generating new instances of the game

**(commands.html)** the game

```
// - an introduction to the game, displayed at the start
```

**End Conditions (end-conditions.html)** displayed on the about screen

**conditions.html)** that provides a new instance of the games overworld

```
// - a callback that provides a new instance of the player
```

**Conditional Descriptions**

**(conditional-conditions.html)** a callback that determines if the game is complete, checked every cycle of the game

**descriptions.html)** a callback that determines if it's game over, checked every cycle of the game

```
var gameCreator = Game.Create(
```

**Frame Builders (frame-builders.html)**

```
"The Beginning",
"Dave awakes to find himself in a cavern...",
```

```
"A very low budget adventure.",
```

```
x => overworldMaker.Make(),
```

```
() => player,
```

```
x => EndCheckResult.NotEnded,
```

```
x => EndCheckResult.NotEnded);
```

```
// begin the execution of the game
```

```
Game.Execute(gameCreator);
```

# Overview

Ganes can be executed either synchronously or asynchronously. Regardless, the **GameExecutor** is responsible for handling all execution of games.

▼ Filter by title

## Synchronous Execution

Getting Started (getting-started.html)

Executing a Game (executing-a-game.html)

Locations

Items (items.html)

+ Characters

Commands (commands.html)

End Conditions (end-conditions.html)

Conditional Descriptions (conditional-descriptions.html)

Frame Builders (frame-builders.html)

## Asynchronous Execution

# Overworld

## Overview

Filter by title

An Overworld is the top level location in a game. A game can only contain a single Overworld. An Overworld can contain multiple Regions.

### Getting Started (getting-started.html)

Overworld

### Executing a Game

### (executing a game.html)

### - Locations

Room  
Room

Region (overworld.html)

Room (region.html)

Room (room.html)

Exit (exit.html)

### Items (items.html)

## Use

### + Characters

An Overworld can be simply instantiated with a name and description.

### Commands

### (commands.html)

```
var overworld = new Overworld("Name", "Description.");
```

### End Conditions (end-conditions.html)

Regions can be added to the Overworld with the **AddRegion** method.

### Conditional Descriptions

### (conditional-

```
overworld.AddRegion(region);
```

### descriptions.html)

### Frame Builders (frame-

Regions can be removed from an Overworld with the **RemoveRegion** method.

### builders.html)

```
overworld.RemoveRegion(region);
```

The Overworld can be traversed with the **Move** method.

```
overworld.Move(region);
```

## OverworldMaker

The OverworldMaker simplifies the creation of the Overworld, when used in conjunction with RegionMakers.

```
var overworldMaker = new OverworldMaker("Name", "Description.", regionMakers);
```

However, the main benefit of using an OverworldMaker is that it allows multiple instances of an Overworld to be created from a single definition of an Overworld.



```
var overworld = overworldMaker.Make();;
```

**Getting Started (getting-started.html)**

**Executing a Game (executing-a-game.html)**

#### - Locations

Overworld (overworld.html)

Region (region.html)

Room (room.html)

Exit (exit.html)

**Items (items.html)**

#### + Characters

**Commands (commands.html)**

**End Conditions (end-conditions.html)**

**Conditional Descriptions (conditional-descriptions.html)**

**Frame Builders (frame-builders.html)**

# Region

## Overview

Filter by title

A Region is the intermediate level location in a game. An Overworld can contain multiple Regions. A Region can contain multiple Rooms.

### Getting Started (getting-started.html)

Overworld

### Executing a Game

### (executing a game.html)

### - Locations

Overworld (overworld.html)

Region (region.html)

Room (room.html)

Exit (exit.html)

A Region presents a 3D space.

+ Characters  
The **x** location always refers to the horizontal axis, with lower values being west and higher values being east.

The **y** location always refers to the vertical axis, with lower values being north and higher values being south.

### Commands (commands.html)

- The **z** location always refers to the depth axis, with lower values being down and higher values being up.

### End Conditions (end-conditions.html)

## Use

### Conditional Descriptions

A Region can be simply instantiated with a name and description.

### (conditional-descriptions.html)

```
var region = new Region("Name", "Description.");
```

### Frame Builders (frame-builders.html)

Rooms can be added to the Region with the **AddRoom** method. The x, y and z location within the Region must be specified.

```
region.AddRoom(room, 0, 0, 0);
```

Rooms can be removed from a Region with the **RemoveRoom** method.

```
region.RemoveRoom(room);
```

The Region can be traversed with the **Move** method.

```
region.Move(Direction.North);
```

The Region can be traversed with the **Move** method.

```
region.Move(Direction.North);
```

## Getting Started (getting-started.html)

The start position, that is the position that the Player will start in when entering a Region, can be specified with **SetStartPosition**.

## Executing a Game

## (executing-a-game.html)

```
region.SetStartPosition(0, 0, 0);
```

## - Locations

## UnlockDoorPair (unlock-door-pair.html)

The **UnlockDoorPair** method can be used to unlock an **Exit** in the current Room, which will also unlock the corresponding Exit in the adjoining **Room**.

## Region (region.html)

```
Room (room.html)
```

```
Exit (exit.html)
```

```
region.UnlockDoorPair(Direction.East);
```

## Items (items.html)

## Characters

Like all other objects, Regions can be assigned custom commands.

## Commands

## (commands.html)

## End Conditions (end-conditions.html)

## Conditional Descriptions

## (conditional-descriptions.html)

```
region.SetStartPosition(0, 0, 0);
```

```
return new Reaction(ReactionResult.OK, "You warped to the start.");
```

## Frame Builders (frame-builders.html)

# RegionMaker

The RegionMaker simplifies the creation of a Region. Rooms are added to the Region with a specified **x**, **y** and **z** position within the Region.

```
var regionMaker = new RegionMaker("Region", "Description.")
{
    [0, 0, 0] = new Room("Room 1", "Description of room 1."),
    [1, 0, 0] = new Room("Room 2", "Description of room 2."),
};
```

The main benefit of using a RegionMaker is that it allows multiple instances of a Region to be created from a single definition of a Region.

```
var region = regionMaker.Make();
```



**Getting Started (getting-started.html)**

**Executing a Game (executing-a-game.html)**

**- Locations**

Overworld (overworld.html)

Region (region.html)

Room (room.html)

Exit (exit.html)

**Items (items.html)**

**+ Characters**

**Commands (commands.html)**

**End Conditions (end-conditions.html)**

**Conditional Descriptions (conditional-descriptions.html)**

**Frame Builders (frame-builders.html)**



# Room

## Overview

▼ Filter by title

A Room is the lowest level location in a game. A Region can contain multiple Rooms.

### Getting Started (getting-started.html)

└─ Region

### Executing a Game

### (executing a game.html)

└─ Room

### - Locations

└─ Region

└─ Overworld (overworld.html)

└─ Region (region.html)

└─ Room (room.html)

└─ Exit (exit.html)

A Room can contain up to six Exits, one for each of the directions **north**, **east**, **south**, **west**, **up** and **down**.

### Items (items.html)

## Use

### Characters

A **Room** is simply instantiated with a name and description.

### (commands.html)

```
var room = new Room("Name", "Description.");
```

### End Conditions (end-conditions.html)

Exits can be added to the Room with the **AddExit** method.

### Conditional Descriptions

### (conditional-

```
room.AddExit(new Exit(Direction.East));
```

### descriptions.html)

### Frame Builders (frame-

Exits can be removed from a Room with the **RemoveExit** method.

### builders.html)

```
region.RemoveExit(exit);
```

Items can be added to the Room with the **AddItem** method.

```
room.AddItem(new Item("Name", "Description."));
```

Items can be removed from a Room with the **RemoveItem** method.

```
region.RemoveItem(item);
```

Characters can be added to the Room with the **AddCharacter** method.

```
room.AddCharacter(new Character("Name", "Description."));
```

Characters can be removed from a Room with the **RemoveCharacter** method.

```
region.RemoveCharacter(character);
```

## Getting Started (getting-started.html)

Rooms can contains custom commands that allow the user to directly interact with the Room.

## Executing a Game

### (executing-a-game.html)

```
room.Commands =  
new CustomCommand(new CommandHelp("Pull lever", "Pull the lever."), true, (game,  
args) =>  
    Overworld (overworld.html)  
    Region (region.html) Exit(Direction.East, true, out var exit);  
    Room (room.html) lock();  
    Exit (exit.html) return new Reaction(ReactionResult.OK, "The exit was unlocked.");  
    })  
Items (items.html)
```

## + Characters

### Commands

#### (commands.html)

### End Conditions (end-conditions.html)

### Conditional Descriptions (conditional-descriptions.html)

### Frame Builders (frame-builders.html)

# Exit

## Overview

Filter by title

An Exit is essentially a connector between two adjoining rooms.

### Getting Started (getting-started.html)

#### Executing a Game

An Exit can be simply instantiated with a direction.

#### (executing-a-game.html)

**Locations**

```
var exit = new Exit(Direction.North);
```

Overworld (overworld.html)

Region (region.html)

An Exit can be hidden from the player by setting its **IsPlayerVisible** property to false, this can be set in the constructor.

Room (room.html)

Exit (exit.html)

#### Items (items.html)

```
var exit = new Exit(Direction.North, false);
```

### + Characters

Or set explicitly

#### Commands

#### (commands.html)

```
exit.IsPlayerVisible = false;
```

#### End Conditions (end-conditions.html)

Optionally, a description of the Exit can be specified.

#### Conditional Descriptions

#### (conditional-

#### descriptions.html)

```
var exit = new Exit(Direction.North, true, new Description("A door covered in ivy."));
```

#### Frame Builders (frame-builders.html)

This will be returned if the player examines the Exit.

Like all Examinable objects, an Exit can be assigned custom commands.

```
exit.Commands =
[
    new CustomCommand(new CommandHelp("Shove", "Shove the door."), true, (game, args) =>
    {
        exit.Unlock();
        return new Reaction(ReactionResult.OK, "The door swung open.");
    })
];
```



**Getting Started (getting-started.html)**

**Executing a Game (executing-a-game.html)**

**- Locations**

Overworld (overworld.html)

Region (region.html)

Room (room.html)

Exit (exit.html)

**Items (items.html)**

**+ Characters**

**Commands (commands.html)**

**End Conditions (end-conditions.html)**

**Conditional Descriptions (conditional-descriptions.html)**

**Frame Builders (frame-builders.html)**

# Item

## Overview

Filter by title

Items can be used to add interactivity with a game. Items can be something that a player can take with them, or they may be static in a Room.

### Getting Started (getting-started.html)

## Use

### Executing a Game

An **Item** is created with a name and description.

### + Locations

```
var sword = new Item("Sword", "A heroes sword.");
```

### Items (items.html)

### + Characters

By default an Item is not takeable and is tied to a Room. If it is takeable this can be specified in the constructor.

### Commands

```
(commands.html)Item("Sword", "A heroes sword.", true);
```

### End Conditions (end-

conditions.html) to another Item. This is useful in situations where the Item changes state. Morphing is invoked with the **Morph** method. The Item that Morph is invoked on takes on the properties of the Item being morphed into.

### Conditional Descriptions

### (conditional-

### descriptions.html)

```
var brokenSword = new Item("Broken Sword", "A broken sword");
```

```
sword.Morph(brokenSword);
```

### Frame Builders (frame-

### builders.html)

Like all Examinable objects, an Item can be assigned custom commands.

```
bomb.Commands =
[
    new CustomCommand(new CommandHelp("Cut wire", "Cut the red wire."), true, (game,
args) =>
    {
        game.Player.Kill();
        return new Reaction(ReactionResult.Fatal, "Boom!");
    })
];
```

## Interaction

Interactions can be set up between different assets in the game. The **InteractionResult** contains the result of the interaction, and allows the game to react to the interaction.

```
var dartsBoard = new Item("Darts board", "A darts board.");
```

```
var dart = new Item("Dart", "A dart")  
{
```

```
    Interaction = item =>
```



```
    {
```

```
        if (item == dartsBoard)
```

**Getting Started (getting-started.html)**

```
            return new InteractionResult(InteractionEffect.SelfContained, item, "The  
            dart stuck in the darts board.");
```

**Executing a Game**

**(executing-a-game.html)**

```
            return new InteractionResult(InteractionEffect.NoEffect, item);
```

```
        }
```

```
    },
```

## + Locations

**Items (items.html)**

## + Characters

**Commands**

**(commands.html)**

**End Conditions (end-**

**conditions.html)**

**Conditional Descriptions**

**(conditional-**

**descriptions.html)**

**Frame Builders (frame-**

**builders.html)**

# PlayableCharacter

## Overview

Filter by title

A PlayableCharacter represents the character that the player plays as throughout the game. Each game has only a single PlayableCharacter.

### Getting Started (getting-started.html)

## Use

### Executing a Game

A PlayableCharacter can be simply instantiated with a name and description.

### (executing-a-game.html)

### + Locations

```
var player = new PlayableCharacter("Ben", "A 39 year old man.");
```

### Items (items.html)

### Characters

A PlayableCharacter can also be instantiated with a list of Items.

```
PlayableCharacter (playable-character.html)
var player = new PlayableCharacter("Ben", "A 39 year old man.",
    NonPlayableCharacter (non-playable-character.html)
    new Item("Guitar", "A PRS Custom 22, in whale blue, of course."),
    new Item("Wallet", "An empty wallet, of course.")
    );
```

### Commands

### (commands.html)

### End Conditions (end-conditions.html)

A PlayableCharacter can be given items with the **AcquireItem** method.

### Conditional Descriptions

```
player.AcquireItem(new Item("Mallet", "A large mallet."));
```

### (conditional-

### descriptions.html)

A PlayableCharacter can lose an item with the **DequireItem** method.

### Frame Builders (frame-

### builders.html)

```
player.DequireItem(mallet);
```

A PlayableCharacter can use an item on another asset:

```
var trapDoor = new Exit(Direction.Down);
var mallet = new Item("Mallet", "A large mallet.");
player.UseItem(mallet, trapDoor);
```

A PlayableCharacter can give an item to a non-playable character.

```
var goblin = new NonPlayableCharacter("Goblin", "A vile goblin.");
var daisy = new Item("Daisy", "A beautiful daisy that is sure to cheer up even the most miserable creature.");
player.Give(daisy, goblin);
```

PlayableCharacters can contains custom commands that allow the user to directly interact with the character or other assets.

```
player.Commands =  
[  
    new CustomCommand(new CommandHelp("Punch wall", "Punch the wall."), true, (game,  
    args) =>  
    {  
        game.Reaction(ReactionResult.OK, "You punched the wall.");  
    }  
]);
```

**Getting Started (getting-started.html)**

**Executing a Game (executing-a-game.html)**

## + Locations

**Items (items.html)**

## - Characters

PlayableCharacter (playable-character.html)

NonPlayableCharacter (non-playable-character.html)

**Commands (commands.html)**

**End Conditions (end-conditions.html)**

**Conditional Descriptions (conditional-descriptions.html)**

**Frame Builders (frame-builders.html)**



# NonPlayableCharacter

## Overview

Filter by title

A NonPlayableCharacter represents any character that the player may meet throughout the game.

### Getting Started (getting-started.html)

**Use**  
A NonPlayableCharacter can be simply instantiated with a name and description.  
(executing-a-game.html)

+ Locations  
`Goblin = new NonPlayableCharacter("Goblin", "A vile goblin.");`

### Items (items.html)

A NonPlayableCharacter can give an item to another NonPlayableCharacter.

### - Characters

```
PlayableCharacter (playable-  
var daisy = new Item("Daisy", "A beautiful daisy that is sure to cheer up even the m  
character.html)  
ost miserable creature.");  
NonPlayableCharacter (non-  
npc.Give(daisy, goblin);  
playable-character.html)
```

### Commands

NonPlayableCharacters can contains custom commands that allow the user to directly interact with the character or other assets.  
(commands.html)

### End Conditions (end-conditions.html)

```
[  
Conditional Descriptions  
(conditional-  
descriptions.html)  
return new Reaction(ReactionResult.OK, "Well that felt weird.");
```

### FrameBuilders (frame-builders.html)

## Conversations

A NonPlayableCharacter can hold a conversation with the player.

- A Conversation contains **Paragraphs**.
- A Paragraph can contain one or more **Responses**.
- A **Response** can contain a delta to shift the conversation by, which will cause the conversation to jump paragraphs by the specified value.
- A **Response** can also contain a callback to perform some action when the player selects that option.

```

goblin.Conversation = new Conversation(
    new Paragraph("This is a the first line."),
    new Paragraph("This is a question.")
    {

```

```

        Responses =

```

```

        [
            new Response("This is the first response." 1),
            new Response("This is the second response.", 2),
            new Response("This is the third response.", 2)
        ]
    }
}

```

## Getting Started (getting-started.html)

## Executing a Game

## (executing-a-game.html)

```

new Paragraph("You picked first response, return to start of conversation.", -
2),

```

## + Locations

```

new Paragraph("You picked second response, return to start of conversation., -
2),

```

## Items (items.html)

```

new Paragraph("You picked third response, you are dead., game => game.Player.Kil
1())

```

## - Characters

```

PlayableCharacter (playable-
character.html)

```

```

NonPlayableCharacter (non-
playable-character.html)

```

## Commands

## (commands.html)

## End Conditions (end-

## conditions.html)

## Conditional Descriptions

## (conditional-

## descriptions.html)

## Frame Builders (frame-

## builders.html)

# Global Commands

## Overview

▼ Filter by title

There are three main types of Command.

**Game Commands** are used to interact with the game.

**Start Commands** are used to interact with the program running the game.

- **Custom Commands** allow developers to add custom commands to the game without having to worry about extended the games interpreters.

**Executing a Game**

(executing-a-game.html)

## Game Commands

**Locations**

**Items (items.html)**

## Drop

Characters

Allows players to drop an item. **R** can be used as a shortcut.

**Commands**

(commands.html)

drop sword

**End Conditions (end-**

**conditions.html)**

The player can also drop **all** items.

**Conditional Descriptions**

(conditional-

drop all

**descriptions.html)**

**Frame Builders (frame-**

**builders.html)**

## Examine

Allows players to examine any asset. **X** can be used as a shortcut.

Examine will examine the current room.

```
examine
```

The player themselves can be examined with **me** or the players name.

```
examine me
```

or

```
examine ben
```

The same is true for Regions, Overworlds, Items and Exits.

# Take

Allows the player to take an Item. **T** can be used as a shortcut.

```
take sword
```



Take **all** allows the player to take all takeables Items in the current Room.

## Getting Started (getting-started.html)

```
take all
```

## Executing a Game (executing-a-game.html)

## Talk

### + Locations

Talk allows the player to start a conversation with a NonPlayableCharacter. **L** can be used as a shortcut.

## Items (items.html)

If only a single NonPlayableCharacter is in the current Room no argument needs to be specified.

### + Characters

## Commands

### (commands.html)

## End Conditions (end

Howe if the current Room contains two or more NonPlayableCharacters then **to** and the NonPlayableCharacter's name must be specified.

### conditions.html)

## Conditional Descriptions

```
talk to dave
```

### (conditional-descriptions.html)

## Use

### Frame Builders (frame-builders.html)

Use allows the player to use the Items that the player has or that are in the current Room.

```
use sword
```

Items can be used on the Player, the Room, an Exit, a NonPlayableCharacter or another Item. The target must be specified with the **on** keyword.

```
use sword on me
```

Or

```
use sword on bush
```

# Move

Regions are traversed with direction commands.

- **North** or **N** moves north.
- **East** or **E** moves east.
- ▼ • **South** or **S** moves south.
- **West** or **W** moves west.
- **Down** or **D** moves down.
- **Up** or **U** moves up.

**Getting Started (getting-started.html)**

## End

**(executing-a-game.html)**

Only valid during a conversation with a `NonPlayableCharacter`, the `End` command will end the conversation.

### + Locations

`end`  
**Items (items.html)**

### + Characters

## Global Commands

**Commands (commands.html)**

## About

**Conditions (end-conditions.html)**

Displays the a screen containing information about the game.

### Conditional Descriptions

`about`  
**(conditional-descriptions.html)**

### Frame Builders (frame-

## CommandsOn / CommandsOff

Toggles the display of the contextual commands on the screen on and off.

`commandson`

Or

`commandsoff`

## Exit

Exit the current game.

`exit`

# Help

Displays a Help screen listing all available commands.

```
help
▼
Key On / Key Off
(key-on-key-off-
started.html)
key on
Executing a Game
(executing-a-game.html)
```

```
+ Locations
Or
Items (items.html)
```

```
+ Characters
key off
Commands
(commands.html)
```

```
Map
End Conditions (end-
conditions.html)
Displays the map screen.
```

```
Conditional Descriptions
(man
conditional-
descriptions.html)
```

```
Frame Builders (frame-
builders.html)
New
Starts a new game.
```

```
new.
```

## Custom Commands

Custom commands can be added to many of the assets, including Room, PlayableCharacter, NonPlayableCharacter, Item and Exit. For more informations see their pages.

# End Conditions

## Overview

▼ Filter by title

The **EndCheck** class allows the game to determine if it has come to an end. Each game has two end conditions

**Getting Started (getting-started.html)**  
The **GameOverCondition** when the game is over, but has not been won.

**CompletionCondition** when the game is over because it has been won.

### Executing a Game

## Use

When **EndCheck** is invoked it returns an **EndCheckResult**. The **EndCheckResult** details the result of the check to see if the game has ended.

**Items (items.html)**

```
+ Characters  
private static EndCheckResult IsGameOver(Game game)
```

```
{  
    Commands
```

```
    if (game.Player.IsAlive)  
        return EndCheckResult.NotEnded;
```

**End Conditions (end-conditions.html)**

```
    return new EndCheckResult(true, "Game Over", "You died!");  
}
```

### Conditional Descriptions

The **GameOverCondition** and **CompletionCondition** are passed in to the game as arguments when a game is created.

**descriptions.html)**

**Frame Builders (frame-builders.html)**

# Conditional Descriptions

## Overview

Filter by title

Normally assets are assigned a **Description** during the constructor. This is what is returned when the asset is examined.

### Getting Started (getting-started.html)

Descriptions are usually specified as a string.

#### Executing a Game

```
var item = new Item("The items name", "The items description.");
```

#### + Locations

They can also be specified as a **Description**.

#### Items (items.html)

```
+ Characters = new Item(new Identifier("The items name"), new Description("The items description."));
```

#### Commands

#### (commands.html)

However, sometimes it may be desirable to have a conditional description that can change based on the state of the asset.

### End Conditions (end-conditions.html)

Conditional descriptions can be specified with **ConditionalDescription** and contain a lambda which determines which one of two strings are returned when the asset is examined.

#### Conditional Descriptions

#### (conditional-

descriptions.html)

```
var player = new PlayableCharacter("Ben", "A man.");
```

#### Frame Builders (frame-

builders.html)

```
var trueString = "A gleaming sword, owned by Ben.";
```

```
// the string to use when the condition is false
```

```
var falseString = "A gleaming sword, without an owner.";
```

```
// a lambda that determines which string is returned
```

```
Condition condition = () => player.FindItem("Sword", out _);
```

```
// the conditional description itself
```

```
var conditionalDescription = new ConditionalDescription(trueString, falseString, condition);
```

```
// create the item with the conditional description
```

```
var sword = new Item(new Identifier("Sword"), conditionalDescription);
```





**Getting Started (getting-started.html)**

**Executing a Game  
(executing-a-game.html)**

**+ Locations**

**Items (items.html)**

**+ Characters**

**Commands  
(commands.html)**

**End Conditions (end-conditions.html)**

**Conditional Descriptions  
(conditional-descriptions.html)**

**Frame Builders (frame-builders.html)**

# Overview

In BP.AdventureFramework output is handled using the **FrameBuilders**. A **FrameBuilder** is essentially a class that builds a **Frame** that can render a specific state in the game. This **Frame** can be rendered on a **TextWriter** by calling its **Render** method. Think of the **FrameBuilder** as the instructions that build the output display and the **Frame** as the output itself.

There are a few types of **FrameBuilder**, each responsible for rendering a specific game state.

## Getting Started (getting-started.html)

- **SceneFrameBuilder** is responsible for building frames that render the scenes in a game.

## Executing a Game (executing-a-game.html)

- **TitleFrameBuilder** is responsible for building the title screen frame.

## Regions (regions.html)

- **RegionMapFrameBuilder** is responsible for building a frame that displays a map of a Region.

- **TransitionFrameBuilder** is responsible for building frames that display transitions.

## + Locations

- **AboutFrameBuilder** is responsible for building a frame to display the about information.

## Items (items.html)

- **HelpFrameBuilder** is responsible for building frames to display the help.

- **GameOverFrameBuilder** is responsible for building a frame to display the game over screen.

## + Characters

- **CompletionFrameBuilder** is responsible for building a frame to display the completion screen.

- **ConversationFrameBuilder** is responsible for building a frame that can render a conversation.

## Commands (commands.html)

A game accepts a **FrameBuilderCollection**. A **FrameBuilderCollection** is a collection of all of the different **FrameBuilders** required to render a game. All **FrameBuilders** are extensible, so the output for all parts of the game can be overridden.

## End Conditions (end-conditions.html)

## Conditional Descriptions (conditional-descriptions.html)

## Frame Builders (frame-builders.html)