GWENT++ Card Compiler

User Documentation

Complete Guide to Creating Custom Cards

Version 1.0
September 25, 2025

Contents

1 Introduction

This compiler allows you to create custom cards for GWENT++ using a Domain-Specific Language (DSL). Write your card and effect definitions in a text file, and the compiler will translate them into a format the game can understand.

2 Defining Effects (`effect`)

Effects are reusable actions that cards can perform. They must be defined before they are used in a card.

2.1 Syntax

```
effect{
    Name: "EffectName",
    Params: {
        parameter1: Type,
        parameter2: Type
    },
    Action: (targets, context) => {
        // Your code here
    }
}
```

2.2 Components

- **`Name`** (String, Required): The unique identifier for the effect.
- **`Params`** (Object, Optional): A list of parameters the effect accepts. The only valid type is `Number`.
- **`Action`** (Function, Required): The logic of the effect. It receives two parameters:
  - `targets`: The list of cards selected by the card's `Selector`.
  - `context`: The game state object.

2.3 Example Effects

```
// An effect with no parameters
effect{
    Name: "Draw",
    Action: (targets, context) => {
        topCard = context.Deck.Pop();
        context.Hand.Push(topCard);
        context.Hand.Shuffle();
    }
}

// An effect with a parameter
effect{
    Name: "Damage",
    Params: {
        amount: Number
    },
    Action: (targets, context) => {
        for (target in targets){
            i = 0;
            while(i++ < amount){
                target.Power -= 1;
            }
        }
    }
}
```

3 Defining Cards (`card`)

Cards are the core elements of the game. They use effects defined in the `effect` blocks.

3.1 Syntax

```
card{
    Name: "CardName",
    Faction: "FactionName",
    Type: "CardType",
    Power: Number,
    Range: ["Range1", "Range2"],
    OnActivation: [
        {
            Effect: EffectDefinition,
            Selector: {
                Source: "sourceName",
                Single: true/false,
                Predicate: (unit) => /* condition */
            },
            PostAction: { /* Optional nested effect */ }
        }
    ]
}
```

3.2 Components

- **`Name`** (String, Required): The card's name.
- **`Faction`** (String, Required): The card's faction (e.g., `"Northern Realms"`).
- **`Type`** (String, Required): The card's type (e.g., `"Oro"`, `"Plata"`, `"Clima"`, `"Líder"`).
- **`Power`** (Number, Required for units): The card's base power. Leader and Weather cards often have `Power: 0`.
- **`Range`** (Array, Required for units): An array of valid ranges. Use `[]` (empty) for cards with no range. Options: `"Melee"`, `"Ranged"`, `"Siege"`.

- `OnActivation` (Array, Required): A list of effects that trigger when the card is played.

3.3 The Effect Definition inside a Card (`EffectDefinition`)

You can reference an effect in two ways:

1. **By Name (Shorthand):** Use a string if the effect needs no parameters.

```
    Effect: "Draw"

```

2. **With Parameters (Full):** Use an object to specify the effect and its parameters.

```
    Effect: {
        Name: "Damage",
        amount: 3 // Parameter value
    }

```

3.4 Selecting Targets (`Selector`)

The `Selector` object defines which cards will be affected by the effect.

- `Source` (String, Required): Where to look for cards.
  - Options: `"board"`, `"hand"`, `"otherHand"`, `"deck"`, `"otherDeck"`, `"field"`, `"otherField"`, `"parent"`.
  - `"parent"` can only be used inside a `PostAction` and refers to the targets selected by the parent effect.
- `Single` (Boolean, Optional, defaults to `false`): If `true`, only the first card matching the predicate is selected.
- `Predicate` (Function, Required): A function that filters cards. It takes a `unit` (card) and returns `true` to select it.
  - **String Concatenation:** Use the `@` operator to join strings within the predicate.

```
        Predicate: (unit) => unit.Faction == "Ro" @ "ma"

```

3.5 Chaining Actions (`PostAction`)

An optional `PostAction` allows you to perform a follow-up effect on the results of the current one. The `PostAction` has the same structure as a primary effect.

- If no `Selector` is provided in the `PostAction`, it uses the same `targets` as its parent effect.
- To filter the parent's targets, use a `Selector` with `Source: "parent"`.

4 Full Card Examples

```
// A simple card that draws another card
card{
    Name: "TestCard1",
    Faction: "Newbies",
    Type: "Oro",
    Power: 5,
    Range: ["Melee", "Siege"],
    OnActivation: [
        {
            Effect: "Draw", // Uses the shorthand
        }
    ]
}
```

Listing 1: Simple card that draws another card

```
// A card that damages all "Roma" faction units on the board by 2
card{
    Name: "TestCard2",
    Faction: "Newbies",
    Type: "Señuelo",
    Power: 0,
    Range: [], // No range for non-unit cards
    OnActivation: [
        {
            Effect: {
                Name: "Damage", // Uses the full definition to pass a
    parameter
                amount: 2,
            },
            Selector: {
                Source: "board",
                Single: false,
                Predicate: (unit) => unit.Faction == "Ro" @ "ma"
            }
        }
    ]
}
```

Listing 2: Card that damages faction units

```
// A complex card that damages "Roma" units, then returns weakened ones to
    the deck, and finally draws a card.
card{
    Name: "TestCard4",
    Faction: "Newbies",
    Type: "Plata",
    Power: 10,
    Range: ["Melee", "Ranged"],
    OnActivation: [
        {
            Effect: {
                Name: "Damage",
                amount: 3,
            },
            Selector: {
                Source: "board",
                Single: false,
                Predicate: (unit) => unit.Faction == "Ro" @ "ma"
            },
            PostAction: { // This effect triggers after the damage is applied
                Effect: "ReturnToDeck",
                Selector: {
                    Source: "parent", // Filters the targets from the parent effect
                    Single: false,
                    Predicate: (unit) => unit.Power < 3 // Returns only weakened units
                }
            }
        },
        {
            Effect: "Draw", // A second, separate effect
        }
    ]
}
```

Listing 3: Complex card with multiple effects

5 Language Features (Inside `Action` and `Predicate`)

- **Operators:** Arithmetic (`+`, `-`, `*`, `/`), comparison (`<`, `>`, `==`), logical (`&&`, `||`), and string concatenation (`@`).
- **Variables:** You can declare and use variables (e.g., `i = 0;`).
- **Loops:** `for...in` loops over lists and `while` loops are supported.
- **Context Properties:** Access game state via `context` (e.g., `context.Hand`, `context.Board`).
- **Card Properties:** Access card data via `card` or `unit` in a predicate (e.g., `card.Power`, `unit.Faction`).

6 Troubleshooting

- Ensure all effects are defined before being used in cards
- Check that all required fields are present in each card definition
- Verify that string concatenation uses the `@` operator correctly
- Make sure `PostAction` selectors using `"parent"` are only used within `PostAction` blocks