# BBFramework for D&D 4e Readme

# FAQ

**What is the BBFramework?**
It's a framework -- a collection of macros -- that simplifies character and monster management, attack and damage rolling, power tracking, and dynamic effects & conditions for Dungeons and Dragons 4th Edition games within Maptool.

**There are other frameworks out there. Why make a new one?**
I made it for my own use in my D&D 4E campaigns, and it's generic enough to be of use to other people I play with. I'm making it available just in case it turns out to be useful to others as well.

**How is the BBFramework different?**
The big advantage of this framework relative to the other (excellent) frameworks out there is that it automatically imports your character data from the `.dnd4e` files created by the online Character Builder utility. This simplifies getting up and running -- when you import your character, your MapTool token is automatically set up with all of its powers, and the attack and damage bonuses calculated by the Character Builder.

**What does it do?**

- It **simplifies remembering** what your powers do. Hovering your mouse over any power gives you a Compendium-esque power description.
- It **tracks dynamic conditions** that apply bonuses to attacks, damage, defenses, saving throws, etc.
- It **imports your character stats and powers** from `.dnd4e` files generated by the online Character Builder, making it easy to import and level up your character..
- It **rolls attack and damage** for all your attack powers, with contextual targeting.
- It **tracks the usage** for your daily and encounter powers, as well as hit points, healing surges remaining, etc.
- It can **provide notifications** of expiring conditions, regeneration, recharge powers, save-ends conditions, ongoing damage, etc. as initiative progresses.

**What does it *not* do?**

- It doesn't automatically apply damage to any other tokens, or deal damage to your token automatically in response to other people's actions (though you could potentially script this yourself).
- It doesn't automatically create game effects for utility powers and item powers, although you can script things with a minimum of effort.
- It doesn't re-do the math done by the Character Builder to come up with your attack rolls, damage rolls and defenses. For instance, unequipping your armor won't automatically affect your AC.

- It doesn't track how many actions you use in your turn, or when your turns begin and end.

**Does everybody in the campaign have to use it?**
No. It's self-contained from the point of view of each character. It doesn't care if one person is using it, or many.

**Can the DM use it for monsters?**
Yes. The DM can use it for monsters too. The DM can use it without players using it; any number of players can use it without the DM using it; or everybody can use it.

**Hey, my power / item didn't get imported correctly! WTF?**
Working from the Character Builder file is a double-edged sword: it allows the framework to automate a huge amount of character creation; however, we also have to live with bugs in the way the CB exports data. See the Known Issues section for more.

**What version of MapTool do I need?**
We run with the latest version: 1.3.b87. I haven't tested compatibility with earlier versions.

**I have a feature request.**
Please leave it in the forum on RPtools.net!

**I have a specific problem with a specific file.**
Please PM me in the forum. I'll try to help you out.

**I want to contribute!**
By all means! Contributions welcomed. PM me for details.

# Tutorial

This section explains how to use the BBFramework for 4e DnD power tracking in MapTool, using the test campaign included in the framework zip.

## 1. Getting started

1. Open the test campaign, located in the **demo campaign/BBFramework demo campaign <version>.cmpgn** file. I recommend using the most recent version of MapTool -- 1.3.b87 or later.
   You should see a Start map with the default grasslands background, populated with some library tokens. For details, see the Components section.
2. Go to the **Real Dungeon** map. You'll see a small party of character tokens (TheFist, Isaac, and Voronwe) and a motley collection of test bad guys.

### 1a. Test-driving as a PC

1. Impersonate the token named **The Fist**. You'll see that the token is pre-set with a few utility macros.
2. In the Impersonate window, find and click the **Refresh All** macro.
   A bunch of frames will open up, showing information about the character and clickable buttons for its powers and items. We'll go through these in detail in the following sections, from simplest to most complex.
   See the Suggested layout section for suggestions on how to lay out the frames.

### 1b. Test-driving as the DM

## Suggested PC frame layout

You can do what you like with the frames, of course, but I like the layout below:
TODO:update


In this layout:

- The Combat and Conditions frames (which are the most used during a combat encounter) are fully visible.
- The Stats frame occupies the dead space at the top of the MapTool window.
- The other frames are auto-hidden on the right, so that they are accessible if needed.

# 3. The Character Frame

This is currently just a record of the choices you have made building your character, and of its stats at the time it was exported from the Character Builder. However, there's no interaction built in yet.

# 4. The Skills Frame

This frame lists all of your character's skills, and lets you roll skill checks and ability checks.

- To do a check for a skill or ability, click its name.or its base bonus value. By default, the check will include your base bonus, plus any activated conditions you have.
- If you need to apply a temporary modifier (like somebody granting you a +1 to the check), or if you need to choose which conditions should apply to the roll, click the **+** icon on the right of the skill or ability button.
- A ‡ denotes your trained skills.
- Skills and abilities that have no dynamic conditions applied to them are shown in grey,

with the base bonus for the check shown in the button.
- If a skill or ability has a bonus or penalty due to a condition, the final value is shown in blue, and the bonus or penalty is automatically applied during the check. Hover the mouse over the value to see details of the bonuses that are applied.

Hover the mouse over the result of a skill check in the chat window to see a breakdown of the roll and the bonuses that were applied.

# 5. The Stats Frame

The Stats frame shows your current HP, temporary HP (if any), ongoing damage and regeneration (if any), current defenses, and current attack and damage modifiers.

- To take damage, enter a value in the text field on the bottom left and press the **–** button.
- To heal, enter a value in the text field and press the **+** button.
- To use a healing surge, press the **+surge** button. If there is a value in the text field, it will be added to your surge value.
- To set a specific value for your HP or Temp HP, click the value itself.
- If you have ongoing damage, the total is shown in orange with a down arrow. Click the **-** button to apply the regeneration to your HP value. Click the value to create a new ongoing damage condition in the Conditions frame. (To remove the condition, remove it from the Conditions window.)
- If you have regeneration, it is shown in yellow with an up arrow. Click the **+** button to apply the regeneration to your HP value. Click the value to set a "temporary" regeneration bonus that is not tied to any conditions.

This frame also shows the current values of your defenses, and any bonuses that you have to attack and defense. These values include any bonuses or penalties granted by your character's active conditions. When you have a bonus to a defense, the new value is shown in blue to remind you that your current value is not at your base value. If you have a bonus condition that is not activated (i.e. that is grey in the Conditions window), the bonus value is not rolled in to the number shown in the panel. Instead, a **+** is appended to the number shown in the panel. Mouse over the value to see all the conditions that have been applied and all conditional bonuses that may also apply.

Click a defense value to set a "temporary" bonus that is not tied to any conditions.

To set a "temporary" bonus to attack or damage that is not tied to any conditions, set the new value in the text fields next to the attack and damage bonus values, and click the **set** button.

See also Conditions below.

# 6. The Inventory Frame

This frame lists all the items that your character possesses in the Character Builder, organized by slot. Equipped items are shown in blue; unequipped items are in grey. If an item has a number in parentheses, it indicates the number of that item that the character possesses (this value is static; it is not tracked as you use ammunition or consumeables).

You can click any item to equip or unequip it. This has the following effects:

- If the item is a weapon or an implement, your character's attack powers will automatically use the newly equipped item (or the unarmed strike, if you unequip all weapons). If you have multiple weapons equipped at the same time, the attack powers will by default use the first weapon listed in the `.dnd4e` file (although you can override this when launching an attack).
- If the item is magical, any additional powers it confers will only be enabled in the Combat frame while the item is equipped. While the item is not equipped, the powers in the Combat frame will be shown as disabled (though you can still use them).

Enabling or disabling other kinds of items (miscellaneous non-magical equipment) has no effect. Equipping and unequipping items currently has no effect on the defense stats shown in the Character Sheet, so, for example, your AC won't change if you unequip your armor. The defense values are pre-calculated by the Character Builder. However, you can respond to pressing item power buttons by writing your own macros on the library token. See also Customizing powers.

# 7. The Combat Frame

The Combat frame lists most of your options for combat encounters.

- At the top, you have your attack powers. When you click these buttons, a generic attack macro is automatically invoked with the attack stats of the power, drawn from the `.dnd4e` file. Damage multipliers, implement damage dice, etc. are all handled transparently. See Attacking below.
- Next, you have utility powers. By default, these just print the flavor and effect of the power to the chat. The main use of these buttons is just to track which powers you've used so far in the encounter. However, you can customize the macros on your library token to do other things when you push the buttons.
- Next, item powers. These are like utility powers, in that they just print effect output to chat and track the usage of the powers. If you have no item daily uses left, any daily item powers that you have not actually used yet will go grey to indicate that they are disabled until the next milestone.
- Hover over any of the above powers to see a summary of the power.

- Diamond icons indicate powers that can be used as immediate reactions or immediate interrupts, or powers with triggers.

## Attacking

Your attack powers are all listed at the top of the Combat frame. To launch an attack:

1. Select the token or tokens that you want to target.
2. Click the name of the attack power you want to use.

By default, the attack will fire one attack roll for each selected token and one single damage roll, using the attack bonus and damage amounts specified in the Character Builder for your currently equipped weapon. It will also include bonuses from any activated conditions. See also Conditions below.

In some cases, you may need to override this default behavior. For instance, if you have the Oath of Enmity power, you can in some circumstances roll an extra attack roll and choose which one you want to use. Or, you may need to apply a bonus to your attack or damage rolls that is not accounted for by the power, like Sneak Attack or Hunter's Quarry. Or, you may want to apply conditional bonuses to the roll.

To override the defaults for an attack power, click the **†** icon on the left of the attack power button. A dialog will open, in which you can add to or edit the attack bonus and damage for the attack, specify the number of rolls to make, select which bonuses should apply, etc.

## Chat output

If someone is curious about the power you just used, they can hover their mouse over the power name in the chat window. It will show the data for that power, as stored in the `.dnd4e` file. (You can turn this off in the user preferences, if you find that the large amount of HTML involved causes lag in your network connection.)

Hover the mouse over any attack or damage roll to see a breakdown of the roll and the bonuses that were applied.

## Modifying power usage and activation

You can modify the count and activation status of any button on the fly. For instance, if you fire a reliable power, it will be marked as being used. However, if it misses, you will want to increment the number of uses remaining.

Click one of the links at the bottom of the frame: increment, decrement, or toggle. The frame will enter a blue "select mode", where you can select which button you want to increment, decrement or toggle. You can also quit the select mode by clicking the "quit select

mode" link at the bottom of the frame.

# 8. The Conditions Frame

The Conditions frame tracks the many kinds of transient dynamic conditions that can affect your character in D&D 4e. These conditions may:

- Give you a temporary bonus or penalty to attack rolls, damage rolls, skill checks, saves, etc.
- Apply a state that affects your options for acting: Restrained, Dazed, etc.
- Give you ongoing damage of various kinds.
- Grant any combination of the above. For example, a potent monster attack could give you a -2 to all your attacks plus 5 ongoing damage, save ends.

You can simulate many of these effects using the Conditions frame.

- Lots of basic conditions whose effects are defined in the D&D rules are pre-set in the framework for you to use out-of-the-box. Use the drop-down list to select the condition you want to apply, and click **Add** to add it to your character's condition list.
- You can create your own conditions by clicking the **+ Add new** button. Use the dialog box to set up the effects granted by the condition.
  Click **Save to character** to create the new condition and add it to your character's condition list.
  By default, the new condition is also saved in a history, so that you can add it back quickly at any time should you need it again. You can prevent this by unchecking the Save in History box in the New Condition dialog box. The history is listed at the end of the Known Conditions list. You can erase selected conditions from the history by clicking **Manage History**.
- Conditions can be active (shown in blue in the list) or inactive (shown in grey in the list). This determines whether the condition's bonuses or penalties are applied by default by the framework. If you have certain conditions that you have to apply frequently to your character, like a temporary bonus granted by an at-will attack that you use often, de-activating the condition lets you keep it in the list for easy toggling on and off, without having it apply to your stats all the time.
- The **x** button removes that condition from the character. If the condition was taken from the framework list or the history, it will not be removed from the list; you can re-apply the condition by selecting it in the drop-down list. If you created the condition yourself, it will be gone. You would have to re-add it from the history, or re-create it from scratch.
- The **»** button opens up the add/modify dialog, where you can modify the properties of the condition.

## Ongoing damage conditions

You can apply an ongoing damage condition in any of the following ways:

- Select “Ongoing damage” from the drop-down list on the Conditions frame, and use the input dialog to set the amounts and types of the ongoing damage.
- Create your own custom condition that includes an ongoing damage component.
- Click the orange button in the Stats window, which lists your current ongoing damage.
- Apply a custom condition created by your DM or another player.

While any condition that has an ongoing damage component is active, the orange button in the Stats window is updated with the total amount of damage you need to take next turn. Click the **-** button next to the orange damage value at the beginning of each of your turns to take your ongoing damage.

If you have multiple conditions that grant ongoing damage of the same type, only the largest value is applied (in accordance with the 4e rules).

## Regeneration conditions

Conditions can apply a regeneration bonus. The sum of all these regeneration bonuses is set in the yellow button in the Stats window. Click the **+** button next to the yellow number at the beginning of each of your turns to add your hit points.

## Bonuses and penalties

A condition can apply any number of bonuses or penalties to different character actions.

- Attacks: When you click an attack power name to run a default attack, all bonuses and penalties for activated conditions will be included in the attack and damage rolls. De-activated conditions will not be included. If you want to change the conditions in effect for a single attack, you can click the **†** icon next to the attack name and check the appropriate boxes in the dialog.
- Defenses: Bonuses to your defenses from activated conditions are always included in the defense numbers shown in the Stats frame. If you have de-activated conditions, they are not rolled into the numbers. Instead, a **+** is shown after the number to indicate that other bonuses may apply. When a defense is affected by a bonus (whether conditional or not), the number is shown in blue. Hover the mouse over the number to see a breakdown of the unconditional bonuses that apply to the number.
- Saves, death saves: When you click either of these buttons, you will be shown a dialog in which you can select the bonuses that should apply. Bonuses from activated conditions are pre-selected; you can also select other de-activated bonuses if they apply to the current roll.

- Skill checks: See the Skills section above.

## States

For each condition, you can specify a MapTool token state that should be set or un-set when the condition is activated or de-activated.

When you add or modify a condition, you can specify the token state in the add/modify dialog.

The preset conditions that are built in to the framework use token states with the same name (if they exist). For instance, the “Prone” condition applies and removes the “Prone” condition from the token when it is activated/de-activated. The only exceptions are ongoing damage states, which look for  “Ongoing Cold”, “Ongoing Fire”, “Ongoing Generic”, etc.

## Durations

Condition durations are not tracked automatically. It is up to you to roll saves for your save-ends conditions, to remove conditions at the beginning or end of your turn or of another character's turn.

However, the framework does try to help you remember timed durations by printing a red **X** in the Conditions frame for timed durations and save-ends conditions.

## Categories

You can set categories for your conditions, which are used to organize the conditions in the Conditions frame.

- You can show and hide categories by clicking their names.
- Conditions drawn from the pre-set list of rules conditions are always put in the "Rules Conditions" category by default (though you can edit them if you want to put them into a new category).

# Importing your own character

## Step 1. Export the .dnd4e file

1. Log into the Character Builder.
2. On the home screen, select your character and click **Export**.
3. Use the browser to set the path and file name of the `.dnd4e` file that will be created.

If the character hasn't been changed in a while, I recommend going into the character and re-saving it before you export it. This seems to update the `.dnd4e` file to the most recent format, which might correct little bugs in the XML format.

## Step 2. Convert the character data

1. Double-click the `CharacterImporter.jar` file in the framework package.
2. Use the first file browser to select the input `.dnd4e` file.
3. Use the second file browser to set the output `.mtmacset` file.

### Command-line alternative

You can run the converter from the command line. Pass the input and output files as additional arguments. For example:

```
java –jar CharacterImporter.jar "path\filename.dnd4e"
"path\filename.mtmacset"
```

### Errors

If the import fails, your character has some kind of item, power or feature that is breaking the conversion. The import process is based on certain assumptions about the structure of the `.dnd4e` file and the tags it contains. These heuristics are based on a small number of samples (i.e. the characters in our campaigns, mostly), and may not be true for all characters. Or, Wizards may have changed the structure of the files since the converter was built. Either way, you're limited in the debugging you can do at this stage. Send me the problematic `.dnd4e` file and I'll take a look.

## Step 3. Set up your tokens

1. Open the test campaign in the framework zip file, which is already set up with a working framework token.
   Alternatively, you can simply drop the `tokens/Lib.BBFramework.rptok` token into your own campaign..
2. Create two new tokens for your character: one token that you will use to control your character in the game, and one library token that will hold your character's macros. Give

them exactly the same name, but prefix the name of the library token by **Lib:**. For example, **Conan** and **Lib:Conan**. The Lib: token must be on the same map as the Lib:BBFramework token. The other PC token can be on any map.

3. "Impersonate" your PC token. Right-click it, and select **Impersonate** from the contextual menu.
4. Import the macro set you generated with the CharacterImporter utility into your character token.
5. Click the **Initialize** macro button on your character token. The framework sorts through your character's data, and creates several groups of new macros on the library token. This may take a few seconds, depending on how much data your character has.
6. You should see frames populated with buttons that represent the character's powers, stats and items. These are the windows you will use to interact with your character during the game.
7. Experiment with the frames and buttons. For details, see the tutorial and the sections below.
8. When the character is ready for use in a campaign, save your character token and library token to `.rptok` files. Ask your DM to import your character's library token into your campaign. If the **Lib:BBFramework** token is not already in the campaign, ask him to import that one too.
   Note that the two library tokens must go in the same map.
9. Add your PC token to the campaign yourself, or ask the DM to do it.
   Make sure that in the game campaign, your character token and its library token have exactly the same name, preceded with **Lib:** on the library token.

Once you have initialized your character token, you may want to remove the **BBCharacterData** macro group that contains the data taken from the `.dnd4e` file. You don't have to do this, but if other players experience lags when you move your token around or use its powers, reducing the amount of data on the token may help.

# Updating characters

If you update a character in the Character Builder to go up a level, add equipment, etc., you must follow all the instructions listed above in the same order. However, before you import your new macro set into your character token, you must delete (at a minimum) the **BBCharacterData** group from the token. This removes the old character data. It will be replaced by the new data read from the new `.mtmacset` file.

When you import the macro set, you will be asked if you want to re-import the other macros too (like the Initialize macro, and the macros for showing the frames). Just click No to avoid duplicating the macros.

If you've retrained out a power or feat, or removed some items from your character, you can also delete their corresponding macros from your library token. You don't have to, but it keeps

the library token a little cleaner.

# Customizing your character

Most of the powers set up by the framework are represented by macros on your library token. This includes all your attack and utility powers, item powers, items themselves, feats, and built-in utilities like healing surges and skill checks. This allows you to easily customize the macros to suit your character.

If you change your character in Character Builder and re-export it, your customizations will be kept (as long as you use the same library token, and do not delete the altered macro from your library token).

## Powers

The framework handles the internal mechanics of usage tracking and attack rolling, but you have lots of scope for customizations to make the powers have game effects.

For instance, say you have a utility like Virtue that lets you spend a healing surge and gain temp HP. The data conversion can't pick up game effects like that automatically. However, you can simulate it by editing the Virtue macro on your library token, and adding two commands: one that decrements the healing surge macro, and one that calls the Temp HP macro and passes it your healing surge value.

- Easy: You can change the flavor text and effect text that is written by default into each power macro.
- Easy-to-advanced: You can add your own custom macro mechanics and rolls to your power macros.
- Moderate: Your power macros can also use the functions described in the table below to interact with the framework. For instance, you can test whether a condition is active in order to determine whether the power should fire some effect or apply some bonus.
- Advanced: Your power macros can alter the arguments passed to them by the framework before those arguments are passed back to the framework in the BBprocessPower function. This lets you customize elements of the power, such as attack values, number of attacks, etc.

### Built-in powers

Built-in powers are passed a json array that contains only the name of the power being run. The name is encoded to prevent problems with quotation marks and special characters; if you want to use it, you should decode it by calling decode(powername).

### Utilities and item powers

Utilities and item powers are passed a json array that contains:

- the name of the power being run. The name is encoded to prevent problems with quotation marks and special characters; if you want to use it, you should decode it by calling decode(powername).
- the data for the power. This data is used by the framework to determine how to process its usage, and to write the mouseover table to chat (if enabled).

## Attack powers

You can customize attacks at two levels:

- By customizing the Generic Attack macro. This macro is called every time an attack is launched.
- By customizing the macro for each specific attack.

Attack powers are passed a json array that contains:

- the name of the power being run. The name is encoded to prevent problems with quotation marks and special characters; if you want to use it, you should decode it by calling decode(powername).
- the data for the power. This data is used by the framework to determine how to process its usage, and to write the mouseover table to chat (if enabled).
- the attack config that defines general stuff about the attack, like the hit and miss effects.
- the weapon stats that define the attack bonus and damage expression that will be used to roll the attack.
- the list of target token names.

## About power usage

Each button is stored with an integer “count” value and a state: usually “Activated” (drawn in blue) or “Default”. The count may be interpreted differently for different types of buttons:

- Combat powers that have a frequency (Encounter, Daily, At-Will) interpret the “count” as the number of uses of that power that are currently available. By default, they are all set to 1. While the count is above 0, the button is colored red, black or green to show the frequency. In addition, if the count is above 1, the current count is shown in the button name (so, if there’s no number shown in the button, you have 1 use of the power). When the count becomes 0, the button is re-colored white to indicate that it is disabled.
- When you use an item daily power, and you have no other item daily uses remaining, the buttons for all other unused item dailies are re-colored in grey, to indicate that they are disabled until the next milestone.
- The grey utility buttons mostly have a “count” of 0, and don’t use the count value. Exceptions are:

    - The Healing Surge button, which shows the number of surges remaining.
    - The Take Ongoing button, which shows the total amount of ongoing damage you have (and is colored in blue when the count is greater than 0).
    - The Death Save button, which shows the number of failed death saves since your last extended rest.Items

The framework calls item macros when the items are equipped or unequipped. Each time, it indicates in the macro.args whether the item is being equipped (1) or unequipped (0). You can therefore respond to this by doing things like activating conditions.

# Custom Buttons

You can add your own custom buttons to the Combat frame, which will call macros with the same name that you write on the Lib token. You could use this approach to launch a combination of powers, or do any kind of custom scripting.

- Create your macro in the group named “Custom Buttons” on your lib token. All macros in this group will be automatically picked up during initialization, and made into buttons on the Combat frame.
- Use the “sort by” field to indicate the usage type for your button. Leave this field blank for a grey button like the other default buttons on the Combat frame (Damage, Healing, etc.), or use Encounter, Daily, or At-Will to have the button colored and its usage tracked accordingly.

The next time you initialize, the Combat window will contain a new section for Custom Buttons that contains your macro.

# Adding custom powers

You can also add custom powers to the Attacks, Utilities and Item Powers groups in the Combat window, but it requires some manual json editing.

Create a new macro on your character token (not the lib token) called AttackData_Custom, UtilitiesData_Custom or ItemPowersData_Custom. In the command, write a new json array that contains an object for each new power you want to add. When you initialize the character, these powers will get rolled into the data in the macro set exported from the Character Builder, listed in the Combat window, and processed exactly like every other power button in the Combat window.

The easiest way to get the structure of the json object you have to create is to copy the array of powers contained in the data macros exported from the CB, paste the array into your new macro, and change the fields as needed for the power you want to create.

It's a good idea to put your custom power data macros in a different group from the macros

exported from the CB. That way, next time you change the character in the CB and need to update your token, you can clear the group that contains the exported data macros without losing your custom macros.

# Functions

The framework defines some useful functions that you can call in your customized macros. Call them as follows:

```
[h:BBhealing(24)]
```
or
```
[h:BBbuttonPush("My Power Name", 2)]
```

| | |
|---|---|
| BBhealing | Heals the character the amount you pass as a parameter.<br>Optional additional parameter: 1 to force an immediate refresh. |
| BBdamage | Deals the amount of damage that you pass as a parameter.<br>Optional additional parameter: 1 to force an immediate refresh. |
| BBhealingSurge | Uses a healing surge and regains hit points equal to the character's surge value.<br>Optional parameter: 1 to force an immediate refresh. |
| BBtempHP | Sets the character's temporary HP to the value you pass as a parameter.<br>Optional additional parameter: 1 to force an immediate refresh. |
| BBneedsRefresh | Marks the panel you specify as needing a refresh. Accepts "Combat," "Conditions," "Defenses," "Inventory," "Character," or "Skills".<br>Use BBrefresh() if you need to actually redraw the panel. |
| BBneedsRefreshAll | Marks all panels as needing refresh. |
| BBrefresh | Redraws all panels that have been marked as needing a refresh by calls to BBneedsRefresh or BBneedsRefreshAll. |
| BBincrementCount | Increases the count of the specified button by one. |
| BBdecrementCount | Decreases the count of the specified button by one. |
| BBgetCount | Retrieves the count of the specified button. |
| BBsetCount | Sets the count of the specified button to the value you specify in the second parameter. |
| BBtoggleState | Toggles the activation state of the specified button. |

| | |
|---|---|
| BBsetState | Sets the activation state of the specified button to the value you specify in the second parameter. Typically “Disabled” or “Default” or “Activated”. |
| BBgetState | Retrieves the activation state of the specified button. Typically will return “Default”, “Disabled”, or “Activated”. |
| BBgetFrequency | Retrieves the usage frequency of the specified button. Typically will return “At-Will”, “Encounter”, “Daily” or “Default”. |
| BBgetType | Retrieves the “type” of the specified button. ? |
| BBbuttonPush | Simulates pushing the specified button. Accepts two arguments:<br>- the button name.<br>- for attack powers, whether or not the configuration window should be opened. 0 for no, 1 for yes. Optional, default is 0.<br>Always ends with a refresh. |
| BBprocessPower | Processes a power. This calls the framework macro, if any, updates the button usage, rolls attacks, etc.<br>Requires one argument: a json array.<br>For built-in powers, the array should contain just the name of the power (encoded).<br>For utility powers and item powers, the array should also contain the power data.<br>For attacks, the array should also contain the attack config, the weapon stats, and the list of targets.<br><br>This function is mostly intended for use within power macros (attacks, utilities, item powers, built-in powers), where the macro is always passed by the framework all the arguments needed for the call to BBprocessPower. In other places, you might find it easier to use BBbuttonPush(), which requires only the name of the button that should be fired. |
| BBtabulatePower | Returns an HTML table that contains a nicely formatted power description. |
| BBtoggleCondition | Toggles the activation status of the specified condition.<br>Optional additional parameter: 1 to force an immediate refresh. |
| BBaddExistingCondition | Adds the specified condition from the framework list to the current character.<br>Optional additional parameter: 1 to force an immediate refresh. |
| BBremoveCondition | Removes the specified condition from the current character’s list.<br>Optional additional parameter: 1 to force an immediate refresh. |

| | |
|---|---|
| BBisConditionActivated | Returns 1 if the token has the condition *and* the condition is activated, or 0 otherwise. |
| BBaddNewCondition | If the specified condition already exists on the token, it is activated. If not, it is added to the token.<br>Requires one argument: a json object that contains the condition data.<br>Optional additional parameter: 1 to force an immediate refresh. |
| BBactivateCondition | Activates the condition. The condition must already exist on the current token. Requires one argument: a json object that contains the condition data.<br>Optional additional parameter: 1 to force an immediate refresh. |
| BBdeactivateCondition | Deactivates the condition. The condition must already exist on the current token. Requires one argument: a json object that contains the condition data.<br>Optional additional parameter: 1 to force an immediate refresh. |
| BBgetDefaultCondition | Retrieves a json object that contains the data for a default blank condition. |
| BBhideCondition | Hides the condition from the Conditions frame, but does not change its activation status. Requires one argument: the name of the condition to be hidden.<br>Optional additional parameter: 1 to force an immediate refresh. |
| BBshowCondition | Shows the specified condition in the Conditions frame, if that condition is already in the character's condition list. Requires one argument: the name of the condition to be shown.<br>Optional additional parameter: 1 to force an immediate refresh. |
| BBtoggleItem | Equips or unequips the item with the specified ID.<br>Optional additional parameter: 1 to force an immediate refresh. |
| BBgetItemDataByID | Gets the json object that contains the data for the item with the specified unique ID value.<br>Note that this unique ID is taken from the Character Builder, and is likely to change when you next export the character, so you should probably not hard-code the value in your argument. Use this function only when you can retrieve the ID from somewhere else (such as in the context of an attack power, where you can retrieve it from the weaponstats object that defines the attack and damage stats for the currently equipped item.) |
| BBgetItemDataByName | Gets the json object that contains the data for the item whose name matches the specified string. Partial matches are accepted: |

| | |
|---|---|
| | e.g. "Dark Deeds" will match "Drowmesh, Armor of Dark Deeds +3" |
| BBgetItemDataBySlot | Gets an array of json objects that contain the data for the items equipped in the specified slot. Note that the Character Builder doesn't recognize all the same categories that are listed in the Inventory frame, such as "Rituals" or "Gear". |
| BBgetDataValue | Gets the value of a tag drawn from the Compendium data for a specified item or power.<br>Accepts two arguments: the JSON object that contains the data for the item or power, and a string that contains the name of the tag whose content should be retrieved (e.g. "Effect" or "Enhancement"). |

# Properties

The framework relies on a number of built-in properties on the PC token and library token, which you can also access and modify in your customizations. For a list, click the Show Properties button on your PC token.

Note that clicking **OK** in the Show Properties dialog will write the variable values in the window back to the token.

# Known issues

## Frame layouts:

Frames sometimes need to update each other. For example, when you add a condition that applies a bonus to one or more defenses, the Condition frame needs to update the Combat and Character frames. However, support for this is a bit limited in MapTool. Basically, only frames that are considered *visible* are updated. For a frame to be “visible”, it must be open and not hidden as a tab behind another frame. However, frames hidden by auto-hide do count as visible. So I’d recommend using the auto-hide feature to get frames out of the way when you don’t need them.

## Data import problems:

The Character Builder `.dnd4e` files have some bugs that affect the characters generated by the framework conversion utility:

- Items that have more than one power have only the first power imported successfully. Subsequent powers are copies of the first.
- Powers with Augment often contain only the first Augment section correctly. Subsequent Augment sections under the same power are copies of the first.

Unfortunately, the ways around this are limited:

- You can alter the data, either in the `.dnd4e` file before generating the macro set for your character, or in the data on your character token after you import the data. However, you will have to re-do the same fix again the next time you import your character.
- You can use custom powers to add the missing item powers. See the customization section above.

# TODO features

- F5 should refresh encounter window also.
- Character sheet when you F12 over a player would be better if it showed dynamic things too, i.e. power usage, etc. F12 should pass the tokenID to the show function, which should do all the looking for the libtoken and all.
- Fancy dialogs to create/modify powers.
- In spawn, check for monster name conflicts
- Separate list for shared conditions, and add a Recent conditions list? Remove "history" and replace it with a "shared" and a "recent" ?
- Update docs & post on rptools
- Find funky powers and items that could break the FW
- Do something about ki focuses and items. Ki focuses work, but you always have to open the attack dialog to use them. Perhaps you should be able to explicitly set a default weapon for each power?
- Make powers with “sustain” and/or “stance” activate when used, and de-activate when clicked again? (Should it create a condition instead? That would allow a reminder to pop for sustain)
- Bonus types to avoid thinking about stacking - ie +2 power bonus to AC
- Damage macro could accept a list of types, and could try to match them automatically against resistances and vulnerabilities.
- Take cover into account automatically -- canSeeToken() - only for players :(
- Add theme to data and display it in character sheet.
- Users should be able to respond to condition activation/deactivation with macro code. Options:
    a. put fields for OnActivate and OnDeactivate inside each condition, which would contain code snippets
    b. call OnConditionActivate and OnConditionDeactivate macros on the lib token, and pass the condition.
    c. let user specify in the condition the names of macros that he wants to call when the condition is activated/deactivated. I like this one best.

  nb. this is now partly implemented, but not yet exposed in the dialog. Unsafe!
- Add a condition like “grants CA,” so it can be shown by a token state?
- Allow conditions to toggle other conditions -- like Invisibility means you also have Combat Advantage. Two items here: 1) “children”: like Invisibility/CA, or Unconscious means you are also Helpless. 2) triggered: like, a condition can make you prone, but it’s a separate condition; you can stand up separately from the rest of the condition. But what if the child condition is already in the list?
- Extend conditions to move speed (prone, slowed, etc)? How?
- Panel layout is basically fixed. Try to be smarter.
- add textfield to inventory for dynamic item management, store/read it in a property and

also persist it in a macro
- Party-level stuff? like a shared loot tracker that anyone can update, party effectiveness bar (taking into account group HP and whether anyone is down, surges remaining, dailies remaining...), plot notes panel
- Better support for Augment (though it’s buggy in the dnd4e files)
- Test powers with primary/secondary attacks. I suspect there will be lots of problems here.

# TODO bugs

- GetItemDataByName doesn't work anymore. It passes an array that contains a single object to GetItemFullName, but GetItemFullName doesn't see the surrounding array. As a result it thinks the json.length is greater than 1, and falls through to the second part of the if condition, and blows up. Figure out what is going on here.