

Vassal Designer’s Guide

Version 3.8.1, 2026-11-13

Credits

VASSAL designed by Rodney Kinney

Designer's Guide: Ed Messina (ed@crucible.cc, mycenae on the VASSAL forums)

Development Team: Rodney Kinney, Joel Uckelman, Brent Easton, Michael Kiefte, Tim McCarron, Brian Reynolds, Yan Lyubashevskiy, Claudio Ciardelli

Testing: Thomas Russ

Website and Forum Management: Ben Smith

Comments on this document and suggestions for improvement are welcomed. Please contact Ed Messina (ed@crucible.cc) with your comments and suggestions.

Programming skills are not necessary for creating fully functional VASSAL modules.

Table of Contents

Overview . . . 7
Preparation . . . 8
- Graphic File Support . . . 10
- Graphic Filenames . . . 11
- Graphic Dimensions . . . 11
- Non-Rectangular Graphics . . . 12
- Performance Impact . . . 12

The Module Editor . . . 14
- Launching the Module Editor . . . 15
- The Module Editor Window . . . 15
- The [Module] Node . . . 18
- Default Module Nodes . . . 20
- Suggested Module Filename Convention . . . 21
- Next Steps . . . 22

Using Properties . . . 23
- Types of Properties . . . 23
- Property Names . . . 23
- Property Values . . . 23
- Comparing a Property to Another Property . . . 26
- Game Piece Properties . . . 27
- Message Formats . . . 28

Maps and Boards . . . 29
- Types of Map Windows . . . 29
- Map Window Attributes . . . 30
- Docked Map Window . . . 31
- Boards . . . 32
- Creating a Map Window . . . 33
- Map Options . . . 33
- Recommended Map Options . . . 34
- Adding Options to a Map . . . 34
- At-Start Stacks . . . 35
- Game Piece Layers . . . 36
- Image Capture Tool . . . 39
- Last Move Highlighter . . . 40
- Line of Sight Thread . . . 40
- Map Shading . . . 42
- Mouseover Stack Viewer . . . 43
- Overview Window . . . 45

Re-center Pieces Button . . . 46
Stacking Options . . . 46
Text Capture Tool . . . 47
Toolbar Menu . . . 48
Zoom Capability . . . 48
Map Grids . . . 49
Hex Grid . . . 49
Rectangular Grid . . . 50
Irregular Grid . . . 50
Multi-Zoned Grid . . . 51
Zone Highlighters . . . 52
Zone Properties . . . 52
Sides . . . 57
Observer Side . . . 58
Next Steps: Restricting Components By Side . . . 58
Game Pieces . . . 60
Game Piece Palette . . . 60
Creating Game Pieces . . . 62
Trait Descriptions . . . 65
Action Button . . . 65
Area of Effect . . . 66
Basic Piece . . . 67
Can Pivot . . . 69
Can Rotate . . . 70
Clone . . . 71
Delete . . . 71
Does Not Stack . . . 71
Dynamic Property . . . 72
Global Hotkey . . . 73
Global Key Command . . . 73
Invisible . . . 74
Layer . . . 75
Mark When Moved . . . 79
Marker . . . 80
Mask . . . 81
Move Fixed Distance . . . 83
Movement Trail . . . 84
Non-Rectangular . . . 85
Place Marker . . . 86
Play Sound . . . 86
Property Sheet . . . 88

Prototype . . . 89
Replace with Other . . . 90
Report Action . . . 92
Restrict Commands . . . 94
Restricted Access . . . 94
Return to Deck . . . 95
Send to Location . . . 96
Set Global Property . . . 97
Spreadsheet . . . 98
Sub-Menu . . . 98
Text Label . . . 99
Trigger Action . . . 101
Prototype Definitions . . . 103
Using Prototypes . . . 104
Pre-Setting Traits in a Prototype . . . 104
Prototype Definitions: Defining a Prototype . . . 104
Game Piece Image Definitions . . . 105
Game Piece Image Elements . . . 105
Game Piece Layouts . . . 107
Game Piece Images . . . 110
Decks and Cards . . . 112
Creating a Deck . . . 112
Deck Properties . . . 115
Creating Cards . . . 115
Cards and Prototypes . . . 116
Copying and Pasting Cards . . . 116
Editing the Contents of a Deck . . . 116
Card Properties . . . 117
Deck Global Key Command (GKC) . . . 117
Card Decks in Practice . . . 118
Generating Random Results . . . 121
Dice Button . . . 121
Symbolic Dice Button . . . 122
Random Text Button . . . 124
Additional Module Components . . . 127
Action Button . . . 127
Charts Window . . . 128
Game Piece Inventory Window . . . 129
Global Key Command (Module Level) . . . 131
Global Options . . . 133
Global Property . . . 134

Map Window Toolbars . . . 136
Multi-Action Button . . . 138
Notes Window . . . 139
Toolbar Menu . . . 140
Turn Counter . . . 141
Pre-Defined Setups . . . 146
About Saved Games . . . 146
The Pre-Defined Setup Menu . . . 147
Help Menu . . . 150
PDF Help File . . . 150
HTML Help File . . . 150
Plain Text Help File . . . 151
About Screen . . . 151
Tutorials . . . 152
Additional Topics . . . 154
Importing Custom Classes . . . 154
Module File Structure . . . 154
File Components . . . 154
Reducing Module File Size . . . 155
Importing an Aide de Camp II Module into VASSAL . . . 156
Translations . . . 156
Translating the VASSAL Engine . . . 158
Creating Module Extensions . . . 159
What an Extension Can Do . . . 159
What an Extension Can't Do . . . 159
Using the Extension Editor . . . 159
Extension Properties . . . 160
Universal Extensions . . . 160
Extension Toolbar Menu Items . . . 161
Extension Prototypes in the Base Module . . . 161
Testing Your Extension . . . 162
Activating an Extension . . . 162
Publishing Your Module . . . 163
File Types . . . 163
Limitations . . . 163
More Information . . . 164
Updating a Module . . . 165
Best Practices . . . 168
Know the Game . . . 168
Don't Try to Enforce the Rules . . . 168
Have All the Required Files on Hand . . . 168

Carefully Target Automation . . . 168
Programmatic Efficiencies . . . 169
Documentation . . . 169
Play a Game! . . . 169
Learn from Others . . . 169
Tutorials . . . 170
Getting Started . . . 170
Sides . . . 170
Maps, Boards, and Grids . . . 171
The Strategic Display . . . 172
Counters . . . 174
Prototypes . . . 177
Dice Button . . . 177
Next Steps . . . 177
Card Game . . . 177

Overview

VASSAL is a free, open-source engine for playing board games by computer. VASSAL supports many types of games, including war games, hobby games, card games, miniatures games, and even board- based role-playing games. You can play opponents live on the VASSAL Server or a peer -to -peer connection, play by email, play by forum, or play offline in hot seat or solitaire modes. There are several hundred VASSAL game modules available for free, and more are being created all the time.

VASSAL is supported on Windows, MacOS, Linux, and other platforms. Thanks to VASSAL'S Java architecture, players on different platforms can play each other without regard to operating system.

VASSAL was originally created in 1999 by Rodney Kinney. The name *VASSAL* comes from its original incarnation as a tool to play online games of the classic *Advanced Squad Leader*, and was originally called *Virtual Advanced Squad Leader*, or *VASL*. VASSAL now extends play to a much wider range of games.

This *VASSAL Designer's Guide* explains how to create game modules and extensions using the Module Editor. It is strongly recommended that you be familiar with the contents of the *VASSAL User's Guide* before attempting to create new modules. This guide covers version 3.8.1 of the VASSAL Engine.

Because of VASSAL's easy to use interface, programming skills are not at all necessary for creating fully functional VASSAL modules. In general, VASSAL modules are not "coded" or "programmed", but *designed*. However, programming skills will definitely be helpful if you plan to create custom functionality or features that the standard VASSAL toolbox cannot handle.

For information on installing VASSAL and using it to play games, consult the *VASSAL User's Guide*.

Preparation

The permutations and combinations possible in board games are nearly infinite: there are board games with a single board, and those with multiple boards; games with many simple pieces and games where the behavior of the pieces is very complex.

As a result, it's up to you to select the proper tools needed for the game you want to make.

Supported Game Types

The world of board games is vast, and VASSAL can accommodate a huge number of games. VASSAL supports any of the following game types:

- Traditional board games such as chess, checkers, *Monopoly*, or *Risk*.
- Hex-and-counter, block, and card-driven war games.
- Hobby games, such as Eurogames.
- Card games (traditional, collectible, or limited).
- Role-playing games that use a tactical map.

This list is not exhaustive; most any type of board-based game could be played on VASSAL.

VASSAL is also an excellent platform on which to playtest new game designs. A game designer has instant access to a worldwide audience of playtesters. No physical game sets need to be printed or distributed. It's easy to add or modify features of the game during the process of game development, and feedback can be obtained in real time.

Live and PBEM Games: One advantage VASSAL has over many Internet board game applications is its support of live play. You can log into the VASSAL server and play opponents in real time. In addition, VASSAL can also be used for Play by Email (PBEM) games. You can even switch between the two for the same game. There are no differences in design between modules played by email and modules played live, although some features may improve game play for one style or the other.

Platforms Supported: VASSAL games can be played on Windows, MacOS and Linux platforms. Further, because of VASSAL's Java architecture, players on different platforms can play against one another without regard to platform. If you have a Mac, and a friend runs Windows, you can play any module against one another.

Scoping a Game

One step that will make building your module much easier is proper preparation. Before you begin the design process, it pays to take time to scope your chosen game. The complete design of a module can take anywhere from a few hours to a few weeks or more. A little planning beforehand can make your module easier to build, easier to create, and easier to maintain later.

At a minimum, every board game has a board and pieces. Everything else is negotiable.

Before even opening VASSAL, some questions to ask include:

- **Rules:** What are the rules of the game?

- **Flow:** What's the basic flow of play in the game? What's the goal of the game? What are the sides?
- **Gameplay Requirements:** Are random results needed for the game? How are these results generated? If dice need to be rolled, what kind of dice are rolled, and how many? Are turns tracked in the game? (For example, in many war games, turns are numbered; but in Monopoly, turns need not be tracked.) Are there limited pieces, or unlimited pieces, or some mix of the two? Is this a tactical game? If so, will players quickly need to determine the range between counters? Will players need a private area for personal possessions, such as cards, tokens, or, units?
- **Graphic Requirements:** What graphics will you use? How many maps does the game need? How will the counter images be generated? Will you need to prepare charts or other play aids?
- **Other Requirements:** Is there any special functionality or rules in the game? Will VASSAL be able to handle them?

Examples of Game Scope

Shown here are some simple examples of game scope. Evaluating the elements of your game in detail will make it easier to determine the required components in your module later.

Preparation: Graphics Files in Your Module

Chess

Chess is played on a single board and has two players. There are 6 kinds of pieces, in two colors, in limited quantities. There are no random results and turns are not tracked. The pieces have no special abilities, but are deleted from the game after being captured.

Small Wargame

A. typical hex-and-counter wargame depicting a single Napoleonic battle may have the following scope:
    - Two players.
    - A single map of the battlefield.
    - Turns are tracked.
    - Six-sided dice are used to resolve battles.
    - Limited pieces—units are placed at game start in fixed locations directly on the game board.
    - On Turn 5, the French player receives limited reinforcements; a place will be needed to keep these reinforcements until they are ready to enter.

Monster Wargame

An ambitious game depicting the entirety of World War II in the Pacific may have this scope:

- There are multiple large maps, depicting several theaters of operation.
- There are multiple sides.
- Players may deploy unlimited pieces, in several unit types and nationalities.

- There are dozens of different pieces available to each side.
- Game money is spent to construct and improve units. The money is in the form of paper certificates that players exchange with the "bank." Players will need a place to keep their unspent money and units before they deploy them.
- Unit counters can be improved through training, or be depleted by damage in combat.
- Leader counters will work differently from unit counters. Instead of being depleted, leaders are killed (removed from the game).
- Turns are tracked, and phases and segments are tracked in each turn.
- The game uses six-sided and ten-sided dice to resolve game results.

Card Game

A card game might have this scope:

- No map image is needed, but a common space is needed to place cards (a "table").
- There are multiple decks, each accessible only to certain players.
- Players will need a place to store their private hands and keep them secret from other players.
- Card decks will be needed: two draw decks and a discard pile. Cards will sometimes need to be facedown or face-up.
- Turns are not recorded, but at the end of each turn, played Cards will be moved to the discard pile. It would be nice to do this automatically.

The possibilities for a game's scope are infinite. As a result, the burden is on you to determine how best to assemble your module, using the tools at hand.

Graphics Files in Your Module

A simple module may have just a few graphic images. A more typical module would require dozens or even hundreds of distinct images.

You need to create, scan, or otherwise acquire the graphics files to be included in your module. Graphic file requirements for a module can include:

- Game boards (for one or more boards)
- Game pieces (for counters, cards, markers, and other game tokens)
- Charts (for tables and game aids)
- Button icons

VASSAL has a limited set of graphics files available for use in building modules. These include a small set of default icons, which you can use for buttons. In addition, you can create a limited set of pieces, using NATO military symbols. See page 69 for more details.

Graphic File Support

VASSAL supports graphic files in SVG, PNG, GIF, and JPG formats. These are listed in order of

preference, with SVG and PNG files being recommended over the other types. SVG and PNG files are the most scalable and reliable, GIF files less so, while using JPG files can cause graphics issues with the display of your module.

Graphic Filenames

When working with graphic files, consider these points:

Unique Names: Even if graphic files come from different locations on your hard drive, once added to your module, they are stored in a common folder. A graphic file added to a module that has the same filename as an existing file will overwrite any existing file. Accordingly, you should make sure all of your graphic files are named uniquely, in order to avoid overwriting existing files.

In some cases, such as when updating a module, overwriting existing files may be desired. See page 110.

Naming Convention: You should establish a standard pattern for graphic filenames. This will help when finding, replacing, or updating your graphics files later on, particularly in modules with many individual files.

For example, in a World War II game, with pieces divided by nationality, division, and unit type, and potentially hundreds of graphic images, you might use this system to help organize the image files:

(3 letter national abbreviation)(Division #)(Unit Type)(Identifier).png.

Examples of resulting filenames from this system could be:

- GerDiv1Inf3.png: A PNG image for German Division 1, Infantry Type 3.
- AmeDiv2Arm4.png: A PNG image for American Division 2, Armor Type 4. Of course, you can decide on any naming convention that fits your module best.

Graphic Dimensions

The dimensions of your graphics are an important factor in determining the performance impact of your module. A module with many sizeable graphics can cause significant performance delays on player systems. In addition, large graphic images can be awkward to manipulate on many computer screens.

While there is no upper size limit to the dimensions of graphics you can use in your module, but for best results, it's suggested you adhere to the following guidelines.

- **Main Boards:** A typical main board is usually 2000-3000 pixels in its longest dimension, and generally under 5000 pixels maximum. If a board graphic must be larger, consider breaking up the board into two or more smaller boards and re-allocating screen real estate. A very large map can be awkward to view on a screen, and will have a major impact on system performance. For example, if the physical game includes a game map, a space for cards, and game tables printed on the map, you could consider moving the card space to a Map Window and the game tables to Chart windows. (In addition, a module can include tools to enable

players to re-scale their view of the map on screen, which can mitigate the limitations of a small map.)

- **Other Boards:** Depending on their purpose, other Boards are usually smaller than the main Board. For example, a Private Window intended to hold a player□s private pieces could be much smaller than the main Board, perhaps 500 pixels across.
- **Pieces:** Pieces, obviously, must be scaled to fit your maps. In particular, if you use a Grid on the map, the pieces must be appropriately scaled for the Grid cells. Most pieces like tokens and counters are between 50-100 pixels across. (Some pieces, like cards or money tokens, are usually larger than ordinary pieces, as they are in physical games. Cards are usually between 200-500 pixels across.)
- **Charts:** Chart graphics are typically from 500-1000 pixels across. (There is no Zoom function for most charts, so for best use, they need to fit easily on most computer screens at full size.)

Preparation: Help and Text Files

- **Icons:** Button icons can be any size. There is no upper or lower limit on dimensions, but 10-50 pixels is probably the most useful size. Test the visual quality of your icons so you can decide on a common, compatible size for your buttons. Icon buttons need not be all the same size, but they should be sized to be easily visible and accessible by your players.

Non-Rectangular Graphics

Most graphics used in games (for example, map, counter, and card images) are rectangular (or square). However, your graphics need not be rectangular if you make use of transparency in creating the files. Both PNG and GIF files support transparency.

For example, to make a circular image for a coin counter, create the coin image as an ordinary, rectangular PNG file in your image editor. Any portion of the image outside of the circular coin portion would need to be marked as transparent using the image editor. When the image is added to a game piece, the counter will look like a circular coin, with no empty space around it.

Game Pieces can include an optional Trait, Non-Rectangular, which can make using non-rectangular graphics easier for players. See page 55 for more information.

Performance Impact

A module is not limited by size on a disk; it is limited by memory space available in RAM only. In general, the graphics used in a module are the biggest driver of memory usage, requiring 4 bytes per pixel for each image that is currently being displayed. Gauge the performance impact of your module accordingly.

For example: A map measuring 2000x3000 pixels is displayed with 20 counters on the map that measure 50x50 pixels each.

The total RAM required equals (2000 x 3000 x 4) + (20 x (50 x 50 x 4)) = 24,200,000 bytes, or approximately 254 MB of RAM.

Help and Text Files

Modules can include any number of help and text files. These files can be used for various purposes, such as:

- To supply help on how to use the module.
- To give credits and acknowledgements for the design of the module.
- To provide rules, rules summaries, or important charts.

Such files will need to be created in the HTML or text file editor of your choice. It□s a good idea to create the necessary files before designing the module. This will make the module design process go more smoothly.

For more information on help files, see page 99.

Additional Tools

The following tools may be useful to have on hand when designing a module:

- **An image editor application:** An image editor will be helpful, to create graphics or manipulate and crop scanned artwork.
- A **text or HTML editor:** A text or HTML editor will be needed for the creation of text or HTML help files.
- **A scanner:** A flatbed scanner is useful for scanning game art, such as maps, counters and cards.
- **A Java compiler (for module developers):** In most cases, and for most games, custom coding will *not* be necessary for the creation of a module, and *no* Java programming skills will be needed. VASSAL is flexible and powerful enough to handle the vast majority of available games without any coding skills. However, a highly automated module may require custom Java coding. In this case, a Java compiler may be necessary for the creation of custom classes. A discussion of such custom coding is beyond the scope of this guide.

The Module Editor

You're ready to begin building a module.

What is a VASSAL Module?

A VASSAL module is simply a set of compressed set of graphic, text, and other files, as well a descriptive file (called a Build File, and stored in XML format) that describes how the other files are expressed in play. A module file has the extension .vmod. There is no limit to the file size for a module, but modules intended to be published to the Vassalengine.org site may be up to 75 MB in size.

All of the files in a module are compressed using ZIP compression. As a result, anyone can examine or retrieve the files that comprise any module by simply unzipping the module, using any ZIP utility. (Unzipping a .vmod file is NOT necessary to play it.)

As VASSAL itself is open-source, so are VASSAL modules. Modules are freely editable in the Module Editor, even ones that have been created already. (VASSAL includes no native way to lock or encrypt a module, or otherwise prevent a module being opened in the Module Editor.)

See Module File Structure*, on page 102, for more information on the BuildFile and other module files.*

Module Creation Overview

VASSAL is designed to be a toolkit for the creation of board games, and like any toolkit, is extremely flexible. As such, it's important to remember that there is no one right way to proceed when creating a module. The process given here is a guideline drawn from experienced designers. Two different designers who design modules for the same game in VASSAL are likely to produce very different results.

In general, a VASSAL module is not "coded" in the traditional sense. Because of the graphic user interface of the Module Editor, module creation does not require any programming skills. Someone with no programming experience can design a simple, functional module in a very short time.

However, VASSAL module creation can entail familiarity with particular programming *concepts* , such as event sequencing, property comparison, manipulation of text strings, unit testing, and other related notions from the world of software development.

In general, the process of creating a new module follows these stages:

I. **Basics:** Create the module and specify the basic settings. Choose the settings (and a board) for at least one map. Save the new module.

AI. **Development:** Create and define the module's other game boards, Game Pieces, and other controls required for game play. Additional controls can include turn counters, map buttons like Zoom and Overview, dice rollers, game charts, scenarios, or other components. (This stage will require the huge majority of development time, and may require multiple saves and restarts of the module to see your changes take effect.)

BI. **Testing:** Test your module. Check gameplay to make sure it works the way you want it to. It's a

good idea to play a few complete games to make sure you haven't left anything out. If not, return to Stage III and adjust what's needed.

IV. **Publication:** Publish your module to the Vassalengine.org web site, or distribute it in some other method.

Using the Module Editor

The main utility for building modules is the Module Editor, an easy -to-use tool with a graphic interface. In the Module Editor, you specify each component, and can assign values, settings, and files to the component.

You install the Module Editor when you install VASSAL, along with the VASSAL Player, the Module Manager, the Extension Editor, and other tools. You'll be using the Module Editor in conjunction with the Module Manager. For complete VASSAL installation instructions and directions on how to use the Module Manager, consult the *VASSAL User's Guide.*

The Module Editor looks and operates the same on all platforms: Windows, MacOS, Linux, and all other platforms.

Launching the Module Editor

To launch the Module Editor, in the Module Manager, select **File | New Module.**

The Module Editor Window

The Module Editor window includes a menu bar, a Toolbar, and Configuration window.

The Menu Bar

The Module Editor menu bar has the following menus:

- **File:** The File menu includes:

o. **New Game:** Starts a new game with the open module.

o. **Load Game:** Loads a saved game or log file.

o. **Save Game:** Saves a game in .vsav format.

o. **Close Game:** Closes the current game.

o. **Begin Logfile:** Begins a log file in .vlog format.

o. **End Logfile:** Closes an open log file in .vlog format.

o. **Save:** Saves the current module.

  o. __ **Save As...:** Saves the current module under a new name.

    - **Edit:** The Edit menu includes **Cut, Copy, Paste, Move, Properties,** and **Translate**. These duplicate the functions from the Configuration window. See *Configuration Window*,

below, for more information.

- **Tools:** The **Tools** menu includes **Create Module Updater** and **Update Saved Games.**
- **Help:** The **Help** menu includes:

p. **Help:** Displays VASSAL HTML help.

o. **User's Guide:** Shows the VASSAL User Guide (in PDF format)

o. **Component Help:** Displays HTML help for the selected module component.

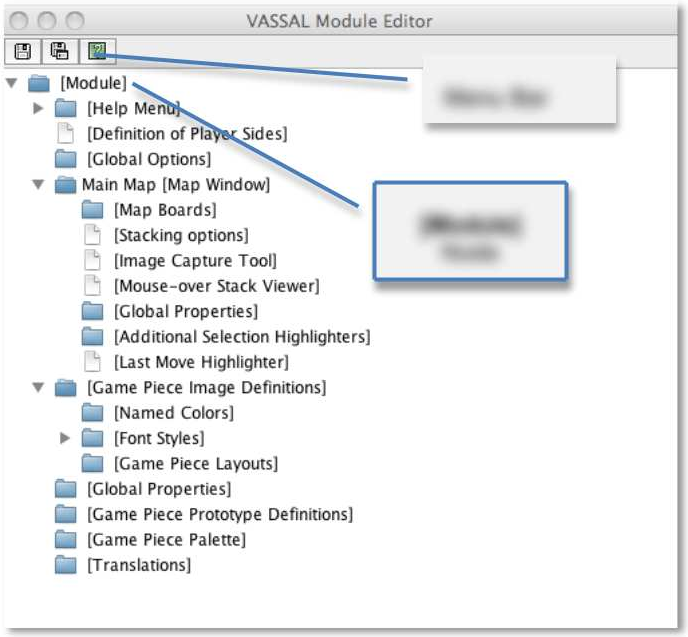
o. **Quick Start:** Displays a short text file for VASSAL newbies.

o. **About Module:** Displays the module splash screen, with information about the module name and version.

The Module Editor Toolbar

The Module Editor Toolbar includes the following buttons:

- **Save, Save As:** Duplicates the items of the same names on the menubar **File** menu. As with any other application, save your work often.
- **Help:** Click **Help** to display the VASSAL HTML help.



Menu Bar

The Configuration Window

Most of the effort of module creation is performed in the Configuration window. Any instructions given here refer to using the Configuration Window to create or configure module components.

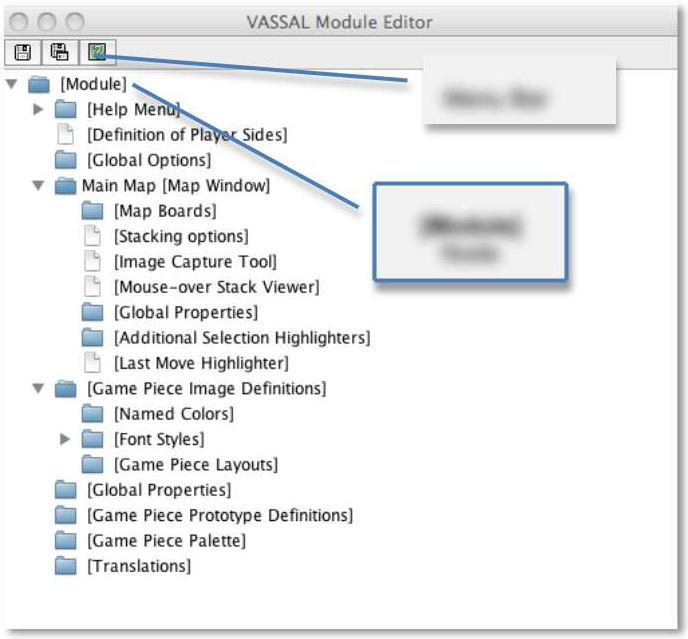
The Configuration window browser displays the module's components as *nodes*, in a hierarchical tree view.

Each node displays a folder icon. Node types appear in brackets **[ ]**. The component name precedes the node type. For example, a node labeled **Japanese Units [Game Piece Palette]** would indicate a

Game Piece Palette component named *Japanese Units*.

Click the arrow next to each folder icon to toggle the expanded folder view and view the various sub -components of the folder. Click the arrow again to contract the node.

You can perform any the following operations on components by right-clicking on the component node and selecting the operation from the menu.



The Module Editor, showing the Configuration Window, Menu Bar, and default nodes for a new module.

- **Properties:** Enables you to choose the settings for the selected component. For components that have already been created, you can access the **Properties** dialog by double-clicking on the selected component.
- **Translate:** Enables you to set translations for the component into a language of your choice. VASSAL is not localized; you must supply the translations for a given module component. See *Translations* on page 104 for more information.
- **Help:** Displays the VASSAL online help for the component.
- **Delete:** Deletes the component. (There is no deletion confirmation prompt, so be careful.) You can also press the **Delete** key on your keyboard.
- **Cut:** Cuts the selected component pasting. A cut and paste will relocate the component. (Alternately, press Ctrl-X on your keyboard.)
- **Copy**: Copies the selected component for pasting. A copy and paste will make a new copy of the component. (Alternatively, press Ctrl-C/Cmd-C on your keyboard to Cut.)
- **Paste:** Pastes a copied or cut component. You can only paste a component to the appropriate place in the tree (like to like). For example, you could copy and paste a Game Piece from one palette to another palette, or to an At-Start Stack, but could not copy and paste the Game Piece to a Turn Counter. (Alternatively, press Ctrl-V/Cmd-V on your keyboard to Paste.)
- **Move:** Moves the component up and down in the tree view. Used to organize and order the components in a logical sequence. (Order of components in the Configuration Window will also

determine the left-to-right Toolbar order of any buttons associated with the components. See page 89 for more information.) After selecting **Move**, you are presented with a dialog to specify a new location for the component in the tree view.

- **Add <Sub-Component>:** Many components include context menu, giving component-specific options, accessible through a right-click. For example, the context menu for a **[Map Window]** component includes a set of options allowing you to add map-specific components, such as a Line of Sight Thread. When created, new sub-components will be shown at the bottom of the list of the node□s sub-components. (Some of these options may themselves have further options.)

The [Module] Node

You create new module components by right-clicking on the **[Module]** node, the topmost node in the Configuration Window. The node is labeled with the module name and contains all the other nodes.

Using the menu from this node, you can create any of the following new components:

- Action Button
- Charts Window
- Dice Button
- Game Piece Inventory Window
- Game Piece Palette
- Game Piece Prototype Definition
- Global Key Command
- Imported Class
- Map Window
- Multi-Action Button
- Notes Window
- Player Hand
- Pre-defined Setup
- Private Window
- Random Text Button
- Symbolic Dice Button
- Toolbar Menu
- Turn Counter

Each of these components is discussed in detail in later sections.

Creating New Components

When creating new components, create just a few of each type of component that you need, and test them first. If you find that you have made a mistake, or that you need to rework pieces or

components, you will not have to go back and correct possibly many examples of the problematic components. For example, if you are creating Game Pieces, create a few Game Pieces first to make sure they function as you intend, and then create the others as needed. (Prototypes can make this process more efficient. See page 67 for more information.)

Copy and Paste

Copy and Paste can be an extremely useful tool when creating or editing a module, as it enables you to create similar components very quickly. Most components in a module can be duplicated by copy and paste. You can then edit the duplicate to create a similar component without having to adjust all the settings.

For example, you may need to create two Map Windows. Each will have similar attributes, differing only in the Board used for each. If you were to create each one individually, you would need to specify the attributes one at a time for each Map Window. However, you could create the first one, adjust the settings and options for the window to what you need, right-click to copy it, and then paste it into the Configuration Window. You could then adjust the settings for the pasted one to individualize it (such as including a new board graphic.) This would save a great deal of time.

The Module Editor will only permit pasting to the appropriate area of the Configuration Window: a Map Window must be pasted into the top-level node of the module, Game Pieces may only be pasted into Game Piece Palettes or At-Start Stacks, and so on.

You cannot cut/copy and paste components between modules.

Creating a New Module

To create a new module,

1. In the Module Manager, select **File | New Module**. The Module Editor opens with a new, empty module with a set of default nodes. In addition, the VASSAL Player loads the game so you can see your changes implemented.

Saving a Module

There are two types of saves.

- **Save:** As with any application, save your work as often as possible. Click the **Save** button in the menu bar to perform a save.
- **Save As:** It□s generally good practice to save renamed copies of your module periodically, as some modifications can be difficult to remove. Use the **Save As** button to save interim copies of your module, under a new filename, before making major edits to your module.

Editing a Module

After a module is created, you can save it at any time, and come back to work on it later

To edit a module,

1. In the Module Manager, select the module you wish to edit and pick **Edit Module**. The Module Editor opens the selected module for editing.

When the Module Editor is open, the VASSAL Player will also load your game in Edit mode. This will enable you to test your module as you create it. Unlike an ordinary game, when in Edit mode, you will not need to log in to the module to test it in the Editor. In the Module Manager, pick **File** | **New Game** to start a game.

Refreshing the Editor

As you make changes to your module, many components you edit will reflect any changes you have made to them in real time.

You will be able to see immediately how the edited component looks or works in the VASSAL Player.

Some modifications, such as new board graphics, sound files, or changes to Prototype Definitions, may not be immediately reflected in the VASSAL Player. As well, the names of some components, such as Game Piece Palette tabs, Charts, and Irregular Grid Regions, may be truncated after you create them. This truncation is merely cosmetic. Any of these additions will require you to re-start the Editor in order for them to be displayed correctly in the VASSAL Player.

As a result, a good habit is to save your work, close, and then re-launch your module after you have made any major changes, particularly after adding or editing graphics files. Click **Save** , and then close the Configuration Window. In the Module Manager, right-click your module and pick **Edit Module** to re-load the module in the Module Editor. Any changes you have made to graphics or components should be fully functional after a restart.

In some instances, you may edit a module but, frustratingly, the changes won□t show up even after you refresh the view. This can occur in games that load a Pre-Defined Setup at game start—changes to a module will not be reflected in a Predefined Setup. See page 110 for more information.

Default Module Nodes

By default, a new module includes the following nodes. Not all of these nodes need be used in a given module.

- **[Module]:** Includes all other nodes, and used to create module-level components.
- **[Help Menu]:** Customize the module help menu.
- **[Definition of Player Sides]:** Define optional player Sides.
- **[Global Options]:** Define global module settings for all players.
- **Main Map [Map Window]:** The default Map Window, which usually contains the game□s main board. May be renamed, modified, or deleted. However, a module will usually include at least one Map Window.
- **[Game Piece Image Definitions]:** Create optional game image layouts.
- **[Global Properties]:** Define optional module-level Properties.
- **[Game Piece Prototype Definitions]:** Define optional module Prototypes.
- **[Game Piece Palette]:** The default Game Piece Palette for generating pieces.
- **[Translations]:** Configure text strings to translate your module.

You can now enter the module□s basic settings.

Module Basic Settings

Module basic settings are displayed for the module□s entry in the Module Manager.

Game Name

Name the module whatever you like. It should correspond to the name of the game. (The module name, which is displayed in the Module Manager, is distinct from the module filename.)

Version Number

Module version number is the number you assign to the current edition of the module. This must be a numeric value. Module Version Number serves these purposes:

- Helps the players to identify the module version they currently are using.
- Acts as a check to make sure that games are created with the same version of the module.
- Ensures the Saved Game Updater Tool can apply attributes from a game created with later version of the module to an earlier version.

Description

The module description is displayed for players in the Module Manager. The description should be brief—no more than a line or so.

To set the module's basic settings,

1. In the Configuration Window, double-click the **[Module]** node. (By default, this is labeled *Unnamed Module*, but the name will change after the module is saved.)
2. In the dialog, enter values for **Game Name, Version Number** and **Description**.
3. Click **Ok**.

Suggested Module Filename Convention

When saving, choose a filename for the module. A suggested filename convention is <game name>_<version number>.vmod.

For example, clue_1.3.vmod, would indicate version 1.3 of a module for *Clue*.

Whatever filename you choose, it □s recommended to always include version number in the filename, so players can quickly tell which version of the game they have without having to open the file.

Some older modules use .zip or .mod as a file extension. However, modules made for VASSAL 3.1 and later must always be given the extension .vmod.

Next Steps

Now you can add other module components, like Map Windows, Game Pieces, and other items. Depending on the scope of your game, some of these components may be optional for your game. See the succeeding chapters for more information on these components.

Using Properties

A *Property* is an attribute of a module component (or the module itself), simply consisting of a name and a value.

Properties are used by VASSAL when executing commands and for determining details of the current game state or components.

Using Properties is an important part of automating your module.

Properties do nothing by themselves. They need to be evaluated by other VASSAL functions or components. Properties can be used as selection criteria for certain pieces, to track game quantities or game events, and for many other purposes.

Types of Properties

Properties come in two types: system Properties, and custom Properties.

- Many VASSAL components, such Game Pieces, Decks, Maps, Boards, Dice Rollers, and Turn Counters, already have Properties defined for them. These are called *System Properties*. System Properties for components are listed in the component descriptions in later sections.
- In addition, you can create and define *custom* Properties for Game Pieces, Map Windows, Zones, and the module itself. Examples of custom Properties include Global Properties, Dynamic Property Traits, and Marker Traits. Information on using component Properties, Traits, or Global Properties will be found in their descriptions in this Guide.

Property Names

System Property Names: The names of System Properties are already defined for module components. For example, all Game Pieces have a system Property named CurrentMap, the value of which is the name the Map where the Game Piece is currently located. System Properties cannot be renamed. System properties for module components are defined in this guide in the sections describing those components.

Custom Property Names: The name of a custom Property you define, such as a Global Property, Dynamic Property Trait, or Marker Trait, must be composed of alphanumeric characters (A- Z, 1-9), and may contain a space ( ). The name cannot contain special characters or punctuation marks. To avoid unpredictable behavior, the name of a custom property should not duplicate the name of a System Property.

Property names are case-sensitive. For example, PowerLevel is not the same Property as powerLevel.

Property Values

System Property Values: System Properties automatically take their values from the game state. You won t need to assign values to System Properties manually (in fact, you cannot do so). Their

values will depend on the condition they describe. For example, the value of the CurrentBoard Property for a Game Piece is the name of the current Board where the piece is located. If you moved the piece to a new Board, the value of CurrentBoard for that piece would change automatically to the name of the new Board.

Custom Property Values: You may assign values to custom Properties. You can assign any of the following types of values:

- *A string of text:* By default, and unless defined otherwise, Properties will accept a string of text as a value. Unless noted otherwise, this is the default type for most Properties. The value of a text Property can include a space ( ) character (for example, *Foo Bar*).
- *Boolean value:* Some Properties are simply checked to see if they are logically true or not. (These are called Booleans.) A Boolean Property will have a text value that can be either "yes/no" or "true/false" (not capitalized).
- *Numerical:* The value of a numerical Property is limited to positive or negative integers (or zero), such as -5, 1, 0, -23, 134, and so on. You can designate some Properties (such as Global and Dynamic Properties) as numerical when you create them.

As with Property Names, Property values are case-sensitive.

Displaying Properties

Game Piece property values are generally invisible to players unless you choose to display them, using a Trait such as Text Label or Layer.

Comparing Properties

You create Property *expressions* to determine if a particular game condition is true. For example, since a System Property named LocationName is used to record a piece's current location, we could check the value of this Property to determine if the piece is currently situated in Hex 1212.

Property expressions are found in many components of a module, and you set them in the dialog boxes for those components when you choose the component settings. Expressions are also sometimes called *filters*, because they filter out situations where the comparison does not apply.

An expression must use one of the following symbols (known as operators) to evaluate the relationship between two values.

Traditionally, a space character ( ) is placed between the Property, the operator, and the value, to improve legibility. However, spaces between them is not required. (CurrentTurn >= 5 is the same expression as CurrentTurn>=5.)

The following table shows each valid operator, the meaning, and under what conditions a comparison using the operator will be true.

Operator	Meaning	True if...
=	Equals	The two values on either side are the same.

!=	Is Not Equal To	The two values are not the same. (The exclamation point (!) is known as a □bang□.)
=~	Regular Expression	The Property value is equal to any one of several values, which are separated by a
|	pipe (|) character.	See *Regular Expressions*, below, for more information.
!~	Regular Expression (Negation)	The Property value is *not* equal to any one of several values, which are separated by a pipe (|) character. See *Regular Expressions*, below, for more information.
>	Greater Than	The value on the left side is larger than the value on the right. Applies to Numerical Properties only.
>=	Greater Than Or Equal To	The value on the left side is larger than or equal to the value on the right. Applies to Numerical Properties only.
<	Less Than	The value on the left side is smaller than the value on the right. Applies to Numerical Properties only.
<=	Less Than Or Equal To	The value on the left side is smaller than or equal to the value on the right. Applies to Numerical Properties only.

Types of Expressions

Typically, expressions are used in a module component to determine the conditions under which the effects of the component should apply. There are several kinds of expressions, which include:

- Simple expressions, which check the Property to see if matches a single value.
- Regular expressions, which check the Property for any of several values.
- Comparing the value of the Property to the value of another Property.
- Indirect comparisons, where one Property name contains the name of another Property.
- Joined comparisons, which can check for multiple conditions.

When creating comparisons, remember that Property names and values are case-sensitive.

Simple Expressions

To check if the value of a Property matches a single value, use a simple expression. For example:

- PieceName = Paratrooper (text)
- CurrentTurn = 10 (numerical)
- ObscuredToOthers = true (Boolean)

In these comparisons, the value on the right side is called a literal*, because the text, number, or condition must be literally true—as written—for the comparison to be true.*

Regular Expressions

A *regular expression* checks if a Property has any one of several values. A regular expression is denoted using the =~ operator. Surround the name of the Property on the left side with $-signs, and separate each value by a pipe character (|). There must be no spaces between pipe-separated values. For example:

 CurrentPlayer =~ Blue|Green|Red (checks if the Blue, Green or Red player is the current player)

You can also negate regular expressions by using !~ instead of ~=.

Comparing a Property to Another Property

On occasion, you may need to compare the value of one Property to the value of another. In this case, surround the name of the Property on the right side of the operator with $-signs (such as $PieceName$) to indicate that the Property with that name should be checked for its value. (Do not use $-signs in the left side of the expression. The left side of the expression is always treated as the name of a Property.) Examples:

- PieceName = $ActivePiece$ (checks if the name of a selected piece is the same as the value of the $ActivePiece$ Global Property.)
- CurrentTurn = $2d6_result$ (checks if the current turn is the same as the random roll of 2 dice.)

In these comparisons, the Property on the right, in $-signs, is called a variable*, because its value may vary.*

Indirect Comparisons

In an indirect comparison, one Property name contains the value of another Property. Set the name of the Property in the left side by using $-signs. For example, if the Property Example has a Property name as a value, then to compare the value of the Property contained in Example to a value, use $ on the left side of the operator.

- $Example$ = 2

Use $ (dollar) signs within the name of a custom Property to indicate that the Property contains the name of another Property. For example, in a game with Red, Green and Blue players, the value of the $PlayerSide$ Property can be *Red*, *Green*, or *Blue*. Using the Send to Location Trait, we want to

send a card to the current active player's private window (each named Red_Home, Green_Home, Blue_Home). For the Trait's destination we could use the Property $PlayerSide$_Home. When evaluated, the value of $PlayerSide$ would be substituted in the string, giving a final value for $PlayerSide$_Home of Red_Home, Green_Home, or Blue_Home.

Joining Expressions

You can check for multiple conditions using AND (&&) as well as OR (||) to join expressions together. For example, to check if a

Game Piece's current board was called Battlefield, *and* that the piece was an Artillery piece, we would evaluate:

CurrentBoard = Battlefield && PieceName = Artillery

- In an AND comparison, both compared Properties must be true for the entire expression to be true.
- In an OR comparison, only one of the compared Properties must be true for the entire expression to be true.

Complex expressions with multiple joins are possible. (Parentheses and brackets are not supported.) Joined expressions are evaluated from left to right, with OR (||) operators evaluated before AND (&&).

For example,

CurrentBoard = HQ || CurrentBoard = Battlefield && PieceName = Artillery || PieceName = Tank

This would evaluate to *true* if the piece were on either the HQ or Battlefield maps, and was either an Artillery or Tank unit. If the piece were on the HQ map, but was an infantry unit, it would evaluate to *false*.

Game Piece Properties

Each Game Piece has its own set of System Properties (each with a name and a value) that can be used for identification by various components.

When looking for the value of a Property of a Game Piece, Global Properties provide the default values. If the Property is not defined on the Game Piece itself, the value will come from a Global Property attached to Zone occupied the by piece, the Map to which it belongs, or the Module overall, in that order.

Traits on a Game Piece search for Properties in the following order:

1. Within each Trait on itself in order from the Trait at the bottom of the list, up to the top Trait.
2. Zone Global Properties defined for the Zone where the Game Piece is currently located.
3. Map Global Properties defined for the Map where the Game Piece is currently located.
4. Global Properties defined at the module level.

A Game Piece cannot directly access:

- Properties on another Game Piece.
- Zone Global Properties on a Zone that the Game Piece is not currently located in.
- Map Global Properties on a map that the Game Piece is not currently located in.

For most components, system Properties are hardcoded as part of the VASSAL engine. However, for Game Pieces, you can create entirely new Properties using the Dynamic Property, Marker, and Property Sheet Traits. See *Game Piece Traits* on page 42 for more information.

Message Formats

Many Traits and module components enable you to customize the message that is displayed to users in the Chat Window when game events take place. A *Message Format* is a formula for creating such a message to players. Message formats are highly customizable and usually include Properties as variables.

For example, the Dice button control includes a message indicating the result of the dice, which is specified in **Report Format.** The default message for the Dice button is **$name$** = **$result$***<$playerName$>. This formula indicates the format of the message to be displayed.

- $name$ is evaluated for the name of the Dice button.
- $result$ is the results of the roll.
- $playerName$ is the name of the player who clicked the button.

If Bill clicked a Dice button named 2d6, and the result was 5, the message displayed in the Chat Window would be: ***2d6*** = *5**<*Bill*>.

Constructing a Message Format

In a Message Format, any word surrounded by $-signs represents a variable, the value of which will be determined when the message is generated during play. When constructing a Message Format for a component, click the **Insert** drop-down menu for a list of available variables for the Message Format. Selecting one of the variables from the menu will insert it at the current cursor position.

Words not surrounded by $-signs will be treated as plain text. This enables you to create plain-language messages using a combination of text and variables.

When a Message Format is used in conjunction with a Game Piece, then any Properties of that Game Piece can be used in the Message Format. See page 44 for more information on Game Piece Properties.

Maps and Boards

A *board* is a playing surface on which pieces are moved. Boards are displayed in *map windows*. By default, a module has one main map window, but is possible to have multiple map windows, and pieces can be moved between them.

Some games have multiple boards, and a single one is selected at game start to play on. Other games include multiple board segments, which are used to build the complete game board at game start. Multiple maps can come in handy to make better use of screen real estate. For example, in a physical game, pieces, cards and other items might all be stored on the main game board. But in a VASSAL module, you can have a separate Map Window for the main board, another specifically for a deck of cards, and others to store players personal items like tokens. You can also add buttons to toggle visibility of these windows, so they can be hidden from view when not in use.

Types of Map Windows

The following types of map windows are available.

Standard Map Window

The standard Map Window holds one or more boards to play on. By default, a new module includes a **[Map Window]** node called *Main Map*, but you can change the name of this default, as well as add any number of new Map Windows.

Private Window

A Private Window works just like a regular Map Window, and has all the same options, but includes an additional control to specify which Sides can access the window. Only the owning Side (or Sides) will be able to access the window. You can further specify whether pieces in the Private Window are visible to other Sides. A Private Window can be used for players to store personal items needed in the game, like units, cards, or money.

Player Hand

A Player Hand is like a Private Window, but specifically intended for use to store a players personal hand of Cards. Items placed in a Player Hand Window will be displayed side by side, horizontally, and will not stack. The owning Side can manipulate Cards (such as turning them face up or face down), and drag new pieces to the Hand. Like a Private Window, you can specify which Sides can access the Player Hand, and whether items in the Player Hand are visible to other players.

To make best use of Private Windows or Player Hands, you will need to add Sides to the game. See page 37 for more information on adding Sides.

Chart

A *Chart* can be defined as a Map Window. This is useful if the players need to interact with the chart in some way, such as to move a tracking piece to record the current turn, player income, or victory points. See *Charts* on page 84 for more information on creating a Map Window as part of a Chart.

Maps can be hidden from view, which can be handy if the pieces on the map are performing some automated function, such as drawing from a Deck of Cards that sends Cards to players automatically. For more on creating hidden maps, see Hiding Toolbar Buttons on page 90.

Map Window Attributes

Each map window may include these settings.

These settings apply to Private Windows and Player Hands only:

- **Belongs to Side:** Click **Add** to add the name of a Side. Only Sides on this list will have access to this Private Window or Player Hand. You may add multiple Sides to the list of owners to enable multiple players to access the window. Once these Sides are set, players may not change the list during a game.
- **Visible to All Players:** If selected, non-owning players will be able to view the contents of Private Window or Player Hand.
- **Use the Same Boards as This Map:** A Private Window or Player Hand may be set to automatically use the same boards as another map window. If left blank, then the Private Window will use its own set of boards.

The rest of these settings apply to all Map Windows, Private Windows, and Player Hands.

- **Map Name:** The name of this Map Window. Each Map Window in the module should have a unique name.
- **Mark Pieces That Move:** If selected, then any Game Pieces with the Mark When Moved Trait will be marked when they are being moved in this Map Window. You can also allow players to set this option in their Preferences.
- **Horizontal/Vertical Padding:** The dimensions of the blank space, in pixels, surrounding the boards in the window.
- **Background Color:** The color to use in the blank space padding.
- **Can Contain Multiple Boards:** If selected, when setting up a new game, the player is prompted for how to arrange the available boards (those assigned to the Map Window) into rows and columns. Useful if the game□s main board is comprised of sections that may be arranged differently for different games.
- **Border Color For Selected Counters:** The color of the border to draw around pieces that have been selected.
- **Border Thickness For Selected Counters:** The thickness of the border, in pixels, drawn around pieces that have been selected.
- **Include Toolbar Button To Show/Hide:** By default, a Map Window is automatically visible when a game begins. However, if this is checked, then this Map Window will not be automatically shown. Instead, a button to show or hide this window will be added to the Main Controls Toolbar. You can specify these settings for the toolbar button:

o. **Toolbar Button Text:** Text of the optional Toolbar button.

- **Toolbar Tooltip Text:** Tooltip of the optional Toolbar button.
- **Toolbar Button Icon:** Icon for the optional Toolbar button.
- **Hotkey:** Keyboard shortcut for the optional Toolbar button.

In VASSAL 3.1.18 and later, entry boxes to specify Toolbar button text, tooltip, icon, and hotkey will not be available until the module is saved, exited, and then reloaded. After exiting and reloading the module, and reopening the map window dialog, you will be able to specify values for these settings.

- **Auto-Report Format For Movement Within This Map:** A Message Format that will be used to report movement of

pieces completely within this Map Window: pieceName is the name of the piece being moved, location is the location to which the piece is being moved (in the format specified above), previousLocation is the location from which the piece is being moved. (Note that this message will only be triggered by drag-and-drop piece movement, but not by the Send to Location Trait.)

- **Auto-Report Format For Movement To This Map:** A Message Format that will be used to report drag-and-drop

movement of pieces to this Map Window from another Map Window: pieceName is the name of the piece being moved, location is the location to which the piece is being moved (in the format specified above), previousLocation is the location from which the piece is being moved, previousMap is the name of the map from which the piece is being moved. (Note that this message will only be triggered by drag-and-drop piece movement, but not by the Send to Location Trait.)

- **Auto-Report Format For Units Created In This Map:** A Message Format that will be used to report pieces that are

dragged to this Map Window directly from a Game Piece Palette: pieceName is the name of the piece being moved, location is the location to which the piece is being moved (in the format specified above).

- **Auto-Report Format For Units Modified On This Map:** A Message Format that will be used to report changes to

pieces on this map: message is the text message reported by the Report Action Trait of the Game Piece being modified.

- **Key Command to Apply to All Units Ending Movement on This Map:** You can specify an optional keyboard shortcut that will be applied to any pieces that are moved on this map. Use this box to force a Game Piece to execute the same command every time it is moved.

Docked Map Window

By default, the first Map Window in the Editor (that is, listed topmost in the Editor window) will be shown docked (attached) to the module main controls and Chat Window. This is usually the Main Map. All other Map Windows will be detached from the toolbar as separate windows.

*Players can control this on an individual basis by de-selecting the **Use Combined Application Window** checkbox, under Preferences, and then restarting VASSAL. Deselecting this will cause all windows to be shown undocked (detached) from the module main controls for that player.*

Boards

Once you've created a Map Window, you must add one or more Boards to it. If you attempt to save a new module without assigning at least one Board, the Module Editor will prompt you to assign one.

The [Map Boards] Node

Some games include multiple boards (or board segments). The beginning of such games consists of either selecting a board to play on, or laying out the board segments for play, sometimes in rows and columns.

If the **Can Contain Multiple Boards** option is checked for the Map Window, and multiple boards are defined for it, a player launching a module is presented with a dialog prompting for a board selection, or for board layout.

If the game includes a random map layout, you may wish to create Map Tiles using the Deck function. See page 77 for more information.

The **[Map Boards]** node settings control the dialog presented for multiple boards. The player is prompted to select the Boards used in the game and their arrangement. (To enable the selection of multiple Boards, when defining the Map Window, select **Can Contain Multiple Boards**.)

If the Map Window only includes a single board, the settings in this node may be ignored.

- **Dialog Title:** The title of the dialog window for choosing boards on this map.
- **"Select Boards" Prompt:** The prompt message in the drop-down menu for selecting boards. (For example: *Choose map sheets for the game.*)
- **Cell Scale Factor:** The relative size of the boards displayed in the dialog compared to their final size during play.
- **Cell Width:** The width of a cell when no board has been selected.
- **Cell Height:** The height of a cell when no board has been selected.
- **Select Default Board Setup:** Click to choose a default set of boards. When a default has been set, the dialog will not be shown to players when a new game is begun. Instead, the game will always be started with the boards you select. If you click this button and then clear the boards, then dialog will again be shown at the start of each game.

Boards

When creating a board, you can choose to define a solid color field of any dimension, or you can use an imported image, such as a scan of a game board.



- **Board Name:** Identifying name of the board.
- **Board Image:** Click **Select** to select a board image.
- **Board Width/Height:** Dimension, in pixels, of the board if no image is used.
- **Background Color:** Color of the board, if no image is used.

Large board image size can have an impact on system performance. See page 8 for more information.

Creating a Map Window

To create a Map Window and one or more boards,

1. Right-click the **[Module]** node and pick **Add Map Window**. The Map Window is added to the Configuration window.
2. In the **Map Window** dialog, specify the window settings.
3. In the Configuration Window, expand the **[Map Window]** node.
4. Right-click the **[Boards]** node and pick **Properties.**
5. In the **Map Boards** dialog, enter the settings for the dialog used to select boards at game start.
6. Right-click the [**Map Boards]** node, and pick **Add Board**.
7. On the **Board** dialog, enter the details of the new map board.
8. Repeat Steps 6-7 for any additional boards as needed.

By default, a module includes a Map Window called *Main Map*. You must perform the above procedure for the Main Map (starting from Step 3) before saving the module.

Map Options

By selecting options for the Map Window, you can customize the behavior of pieces on it. By selecting different options for different maps, the same piece may behave differently when on those maps.

Customize a Map Window with any of the options listed here. Each new option added to a Map Window will create a corresponding node with its own settings.

- Additional Selection Highlighter
- At-Start Stack
- Game Piece Layers
- Global Key Command

- Hide Pieces Button
- Image Capture Tool
- Last Move Highlighter
- Line of Sight Thread
- Map Shading
- Mouseover Stack Viewer
- Overview Window
- Re-center Pieces Button
- Stacking Options
- Text Capture Tool
- Toolbar Menu
- Zoom Capability

Default Nodes: A newly created Map Window includes these nodes by default: **[Stacking Options]**, **[Image Capture Tool]**, **[Mouseover Stack Viewer]**, **[Global Properties]**, **[Additional Selection Highlighters]**, and **[Last Move Highlighter].** You can configure these nodes, delete unneeded ones, or freely add new ones to the Map Window.

Recommended Map Options

Although all Map Options have their uses, always consider adding these visibility options to each Map:

- **Mouseover Stack Viewer:** (see page 28) Enables viewing of the contents of a stack of pieces.
- **Show/Hide Pieces:** (see page 26) Enables players to toggle piece visibility, to view the map directly without moving or interfering with pieces.
- **Zoom Capability:** (see page 31) Enables re-scaling of the Map, for easier viewing.

Adding Options to a Map

To add an options node to a Map Window,

1. Right-click the selected **[Map Window]** node, and select an option to add from the context menu.
2. As the option is added, a dialog box is shown. Specify the option settings in the dialog box.
3. Repeat Steps 1-2 until all desired options are added.



Additional Selection Highlighter

An Additional Selection Highlighter enables you to define additional Tpways to highlight the selected piece on a map. The additional highlighters are drawn only if the selected piece matches the specified Properties. If a Game Piece matches the

Properties of more than one highlighter, all will be drawn, in addition to the highlighting color/border specified in the Map's Properties.

An Additional Selection Highlighter has these attributes:

- **Name:** Short name of the component.
- **Active if Properties Match:** The highlighter will be drawn for all Game Pieces on the map that match the given Property expression.
- **Use Image:** Specify an optional image to be overlaid on top of the selected piece. The center of the image will be offset from the center of the piece by the given number of pixels.
- **Border Color:** The color of the border to be drawn around selected pieces.
- **Border Thickness:** The thickness of the border.

At-Start Stacks

An At-Start Stack is a stack of playing pieces that is automatically placed at the beginning of every game. Once the game begins, the pieces will be in place just as if they had been dragged from the Game Piece Palette.

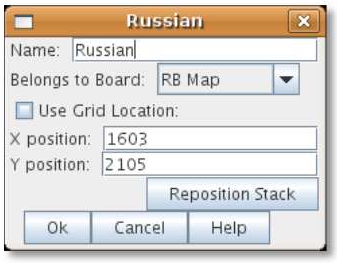
First define the name, map, and position of the At-Start Stack, and then create the individual pieces in the Stack. (You can cut and paste pieces to an At-Start Stack from a Game Piece Palette, or other At-Start Stack.)

An At-Start Stack could be used for the following:

- Any group of Game Pieces whose quantity is fixed (for example, the number of houses in a *Monopoly* set).
- Game Pieces which are found in the same place on the board at the beginning of *every* game (and every game scenario). If the starting pieces or their positions will vary based on the scenario, use a Pre-Defined Setup instead. (See page 97 for more information on Pre-Defined Setups.)

An At-Start Stack should only include the pieces at a given starting location. For example, chess pieces start in 32 locations on the board, and so would require 32 different At-Start Stacks, each consisting of 1 piece each.

If Game Pieces are to be drawn randomly from a selection of pieces, use a Deck instead of an At-Start Stack. See page 74 for more information on Decks.



An At-Start Stack has these attributes:

- **Name:** Identifying name of the stack. (Not used during play.)
- **Belongs to Board:** If a name is selected, the stack will appear on that particular Board. If a game does not use that Board, then the stack will not appear. If *Any* is selected, then the stack will always appear at the given position, regardless of the boards in use.
- **Use Grid Location:** If selected, you can enter the position of the stack using a descriptive location name. This can be the name of a grid point or cell number (for example, on a hex grid, 1515 would place the stack in hex 1515.) Otherwise, you must specify X and Y coordinates.
- **X, Y position:** The position in the Map Window of the center of the Deck. If this stack belongs to a Board, the position is relative to the Board's position in the Map Window.
- **Location:** The location of the stack as a descriptive location label as returned by Grid Numbering or the name of a Region. The Grid numbering system must provide enough information to define a specific location on the map (for example, $GridLocation$). However, if a zone in a Multi-zone Grid does not specify a Grid, the center of the zone will be selected.

EXAMPLE: A strategic game in which a nationality has a fixed force pool of Infantry and Armor counters can be modeled by making a Map Window representing the force pool, with an At-Start Stack of Infantry counters and an At-Start Stack of Armor counters.

Editing the Contents of an At-Start Stack

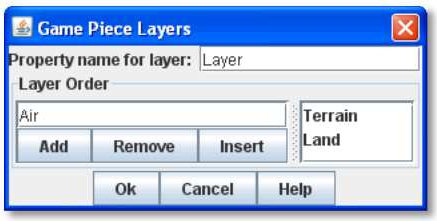
You can make wholesale changes quickly to the entire contents of an At-Start Stack in the Editor. In the Configuration Window, right-click the **[At-Start Stack]** node and pick **Edit All Contained Pieces**. The **Properties** dialog for the first piece is displayed, but any changes you make in the **Properties** dialog will affect all Game Pieces in the At-Start Stack. Add, remove or edit Traits as usual, then click **Ok**. Your changes are applied to all Pieces in the At-Start Stack.

Game Piece Layers

Using Game Piece Layers (GPLs) enables you to specify that certain Game Pieces will always be drawn on top of others. GPLs function like a set of transparent sheets, laid in ascending or descending levels. Pieces on one of the levels will not stack with pieces drawn on other levels above or below it.

After defining the GPLs for a Map, you need to use a Marker Trait to assign each Game Piece (or Prototype) to a GPL. Pieces with no value for the Marker Trait will be drawn on the topmost layer.

See page 52 for more information on assigning a Game Piece to a Game Piece Layer.



The GPL option has these settings:

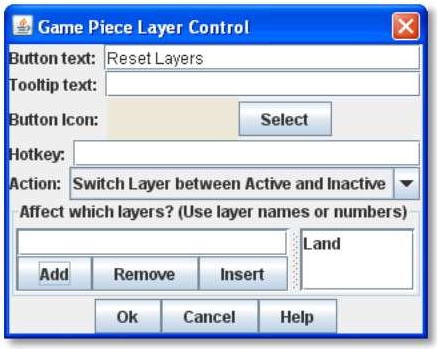
- **Property Name for Layer:** Property name for the Marker Trait used to identify the piece's GPL. The default value is *Layer*.
- **Layer Order:** Click **Add** to specify the Layer order. Each corresponds to the piece's value for the Marker Trait used to identify the GPL. Layers are shown in inverted order from their layout on the map; that is, layers shown at the top of the list are drawn below the ones after them.

Example: A Map has a Game Piece Layer specified with Property name Layer *and Layer Order* Terrain, Land, Air. *Any piece with a Marker Trait with Property name* Layer *and value* Terrain *will be in the bottom-most layer. The middle layer will contain pieces with the value* Land, *and the top layer will contain pieces with the value* Air. *Pieces with no value for the Layer Property will be in their own layer, above all three.*

The Game Piece Layer Map option is not related to the Layer Trait for Game Pieces. See page 49 for more information on the Layer Trait.

Game Piece Layer Control

The Game Piece Layer Control adds a button to the Map Window Toolbar that enables you to activate or deactivate the Game Piece Layers for that map, and to change their relative order. Game Pieces belonging



to GPLs that have been deactivated are hidden from view until the Layer is activated again.

Each player can activate or deactivate Layers independently, and layer activation is not saved when the game is saved.

The Game Piece Layer Control has these settings:

- **Button Text:** Text label for the GPL Control button.

- **Tooltip Text:** Tooltip text displayed on mouseover.
- **Button Icon:** Icon used for the GPL Control button.
- **Hotkey:** Keyboard shortcut for the button.
- **Action:** Action taken when the button is clicked. Choose one of the following:

o. *Rotate Layer Order Up/Down* will change the relative order of the Layers on the map, moving each layer up or down by one in the order.

o. *Make Layer Active/Inactive* will activate or deactivate the specified Layers.

o. *Switch Layer between Active and Inactive* will toggle the specified layers between active and inactive.

o. *Reset All Layers* makes all Layers active and restores them to their default order.

Global Key Command (Map Window Level)

The Global Key Command (GKC) adds a button to the Map Window Toolbar. Clicking the button will select certain pieces in the Map Window and apply the same keyboard command to all of them simultaneously.

By default, a Global Key Command assigned to a Map Window will only affect pieces in the Map Window to which it is assigned. You can specify a new map window by including a CurrentMap expression in **Matching Properties**, which will override the default window. (For a GKC that will affect pieces on any map, use the GKC (Module Level) control, described on page 87.)

The Global Key Command has these settings:

- **Description:** A description of the action, used for the button's mouseover tooltip.
- **Key Command:** The keyboard command that will be applied to the selected pieces.
- **Matching Properties:** The command will apply to all pieces on the map that match the given Property expression.
- **Within a Deck, Apply To:** Select how this command applies to pieces that are contained within a Deck.



o. *No pieces* means that pieces in a Deck ignore the command.

- *All pieces* means that the command applies to the entire Deck.
  - _ *Fixed number of pieces* enables you to specify the number of pieces (drawn from the top) that the command will apply to.
    - **Tooltip text:** Mouseover hint text for the Toolbar button.
    - **Button Text:** Text for the Toolbar button.
    - **Button Icon:** Icon for the Toolbar button.
    - **Hotkey:** Keyboard shortcut for the Toolbar button.
    - **Suppress Individual Reports:** If selected, then any auto-reporting of the action by individual pieces by the Report Action Trait will be suppressed.
    - **Report Format:** A Message Format that will be echoed to the Chat window when the button is pressed.

Commands applied by Global Key Commands will be affected by piece ownership. If the GKC triggers a command that is restricted by side, the action may not take place as intended when the restricted side triggers the GKC (by button or other command).

Hide Pieces Button

Clicking a Hide Pieces button will temporarily hide all pieces on the map from the clicking player, until the button is clicked again. This is useful to get a better look at the game board, such as to read a map label, terrain hex, or legend. (To make pieces invisible to other players, use the Invisible Trait.)

The Hide Pieces Button has these settings:



- **Button Text:** The text of the **Hide Pieces** button to be added to the Toolbar.
- **Tooltip Text:** Text shown on mouseover.
- **Hotkey:** Keyboard shortcut for toggling hidden pieces.
- **Icon When Pieces are Showing:** Button shown when pieces are visible.
- **Icon When Pieces are Hidden:** Button shown when pieces are hidden.

If possible, use a different button image for the showing and hidden icons. Players will be able to more clearly determine when the button has been clicked and when pieces are hidden from view.

Image Capture Tool

The Image Capture tool component adds a button to the Toolbar of the Map Window. Clicking the

button will copy the contents of the Map Window to a PNG image file. Using the Image Capture Tool, you can take an image of the entire map, shot even if the Map Window is too large to fit entirely on the screen.



The Image Capture Tool has these settings:

- **Button Text:** Text label for the Image Capture button.
- **Tooltip Text:** Tooltip text displayed on mouseover.
- **Button Icon:** Icon used for the Image Capture button.

Maps and Boards: Map Options

- **Hotkey:** Keyboard shortcut for the button.

Last Move Highlighter

A Last Move Highlighter draws a colored border around the last piece to have been moved, added, or deleted in a logfile or by an opponent during live play. Clicking on the map clears the highlight.

The Last Move Highlighter has these settings:

- **Enabled:** Enabled by default. If selected, the highlighter is in effect for the last piece to be moved, added, or deleted from a logfile and live play.
- **Color:** Color of the border shown.
- **Thickness:** Border thickness, in pixels.



Line of Sight Thread

A Line of Sight Thread adds a button to the Toolbar of the Map Window. Clicking the button will enable a player to drag the mouse cursor between any two points in the Map Window, drawing a line between those two points to indicate line of sight or range.

The Line of Sight Thread has these settings:

- **Button Text:** The label on the button in the Map Window Toolbar.

- **Tooltip Text:** Tooltip text for the button in the Map Window Toolbar.
- **Button Icon:** Icon for the button in the Map Window Toolbar.
- **Hotkey:** Specifies a keyboard shortcut for the button.
- **Report Format:** A Message Format that specifies the report to the chat window when the LOS button is used. If blank, no report is sent to the chat window when drawing a thread.
- **Persistence:** Select one of the following for the persistence of the LOS thread.

o. *Ctrl-Click & Drag:* The thread will only persist when the drawing player holds down Ctrl-Click and draws the thread.

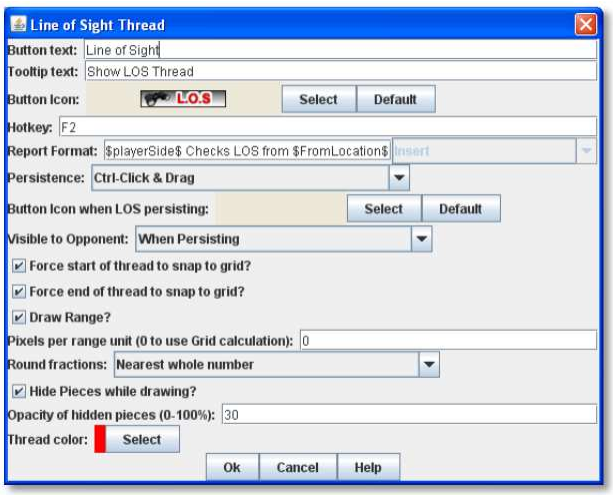
o. *Never:* The thread will only persist as long as the drawing player□s finger is on the mouse button.

o. *_Always:* The thread will persist on the board until a new thread is drawn.

- **Button Icon When LOS Persisting:** The button icon shown when the LOS thread is persisting, in the circumstances defined under **Persistence**.
- **Visible to Opponent:** Select whether a drawn thread will

be visible to the opponent: *When Persisting, Never, Always*.

- **Force Start of Thread to Snap to Grid:** If selected, and a Grid is defined for the map, the thread will always begin in the center of a Grid cell.
- **Force End of Thread to Snap to Grid:** If selected, and a Grid is defined for the map, the thread will always end in the center of a Grid cell.
- **Draw Range:** If selected, draws the range between the two points, in hexes or squares, as appropriate for the board in use.
- **Pixels Per Range Unit:** If drawing the range on a board without a Grid, this determines how many pixels on the screen equal a single unit of range.
- **Round Fractions:** For distances that are a fraction of a range unit, specify whether to round fractions up, down, or to the nearest whole number.
- **Hide Pieces While Drawing:** If selected, then all Game Pieces in the map will be hidden (or transparent) while the thread is being drawn.



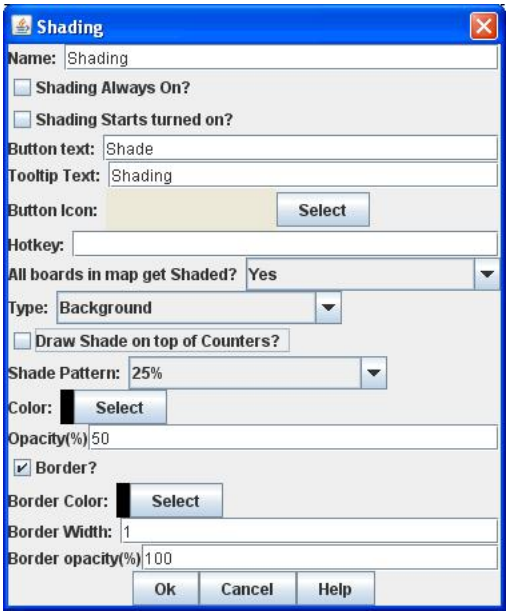
- **Opacity Of Hidden Pieces:** Set the transparency of Game Pieces, as a percentage of original opacity, while the thread is being drawn. 0 is completely invisible, 100 is completely opaque.
- **Thread Color:** Specifies the color the thread on the screen. If set to null (by clicking the **Select** button and then the **Cancel** button in the color-choosing dialog), then a Preferences option will determine the color of the thread at game time.

Map Shading

The Map Shading option applies a semi-transparent solid color or image tiling to the Map. In background mode, Map Shading can be used to overlay a repeating image over solid-color boards. In foreground mode, the area is determined by the pieces on the map that name this Map Shading in an Area of Effect Trait.

The Map Shading option has these settings:

- **Name:** A short name of this shading for reference by pieces with the Area of Effect Trait.
- **Shading Always On:** If selected, then the shading is always drawn. If not selected, then visibility is controlled by a button in the Map Window Toolbar.
- **Shading Starts Turned On:** If selected, then the shading will begin visible when a game is loaded.
- **Button Text:** Text for the Toolbar button.
- **Button Icon:** Icon for the Toolbar button.
- **Hotkey:** Keyboard shortcut for the Toolbar button.
- **All Boards In Map Get Shaded:** Allows you to select which Boards in the map to apply the shading to.
- **Type:** If set to *Background* then the shaded area includes the entire board, minus the areas attached to any Area of Effect Traits. If set to *Foreground*, then the shaded area includes only the areas attached to Area of Effect Traits.
- **Draw Shade On Top Of Counters:** If selected, then the shading will be drawn over any counters on the map. Otherwise, it will be drawn underneath all counters.
- **Shade Pattern:** Choose between 100/75/50/25% hatch patterns, or choose a custom image.
- **Color:** The color of the shading (if not using a custom image).
- **Opacity:** The opacity of the shading. 0 is invisible, 100 is completely opaque.
- **Border:** If selected, will draw a border around the shading area. You can specify the thickness, color, and opacity of the border.



Mouseover Stack Viewer

A Mouseover Stack Viewer displays the contents of a stack when a mouse cursor is moved over it, after a specified delay. The Viewer can also display descriptive text about the pieces in the stack. (Note that a □stack□ can consist of a single piece or multiple pieces.)

The option has these settings:

- **Recommended Delay Before Display:** When the mouse has been stationary for this many milliseconds, the viewer will appear. (Individual users can override this by choosing a setting in **Preferences**. See the *VASSAL User□s Guide* for more information on setting Preferences.)
- **Keyboard Shortcut to Display:** Players may display the viewer without waiting by typing this keyboard shortcut. This can be disabled in the preferences.
- **Background Color:** Pieces and text are drawn against a background of this color.
- **Border/Text Color:** Color of any text drawn, and the border around the overall viewer.
- **Display When At Least This Many Pieces Will Be Included:** Minimum number of units in a stack that will trigger the viewer. You can set this to 1 to view individual pieces. If set to 0, then the viewer will display even if the location is empty.
- **Always Display When Zoom Level Is Less Than:** Regardless of the above **Display When At Least This Many...** setting, the viewer will also display when the map's Zoom level is less than this number.
- **Draw Pieces:** If selected, then the stacked pieces will be depicted in the viewer.
- **Draw Pieces Using Zoom Factor:** The magnification factor to use to draw the pieces in the viewer.
- **Width Of Gap Between Pieces:** Empty space in pixels to place between each drawn piece.
- **Display Text:** If selected, then the viewer will show summary text and some individualized text for each piece. If selected, specify each of these values:

o. **Font Size:** Size of the text shown in the viewer.

o. _ **Summary Text Above Pieces:** A Message Format specifying the text to display above the drawn pieces in the viewer. By default, this is set to $LocationName$. In addition to standard Properties, you can include a Property with the name $sum(PropertyName)$ where (PropertyName) is a Property defined on a Game Piece. The numeric values of this Property for all included pieces will be substituted.

- **Text Below Each Piece:** A Message Format specifying the text to display below each included piece.
- **Include Individual Pieces:** Specifies how pieces are to be selected for inclusion in the viewer. You may restrict the pieces according to the Game Piece Layer that they belong. Alternatively, you may specify the value of a Property.
- **Include Non-Stacking Pieces:** If selected, then non-stacking pieces are eligible for inclusion in the viewer.
- **Show Pieces In Unrotated State:** If selected, then pieces that can rotate are drawn in the mouseover as they look when not rotated.
- **Include Top Piece In Deck:** If selected, then the top piece of a Deck will be shown in the Viewer.

"Offboard" Pieces

By default, a Mouseover Stack Viewer will display each stack showing the value of each piece's current location above each piece. If no Grid is defined for the map, the pieces will be shown as "offboard".

To change the display of the word "offboard", do one of the following:

- Add a Grid to the map. The Viewer will display the stack's current location.
- In the **Mouseover Stack Viewer** dialog, select **Display Text**. In **Summary Text Above Pieces**, delete the Property name $LocationName$.
- As above, but instead of $LocationName$, substitute the name of a different Game Piece Property to be displayed.

Showing the Number of Items in a Stack

You can set a Stack Viewer to show the number of items contained in a stack.

1. Set a Marker Trait on all units you want to count. Name the Marker Trait *UnitCount*, and set the Value to 1.
2. Create a Stack Viewer for the Map Window. In **Summary Text Above Pieces**, select *$sum(PropertyName)$*. In the box, replace *PropertyName* with *UnitCount* (so it shows *$sum(UnitCount)$*). On mouseover, the Viewer will now display the total Unit Count of all pieces in the stack.

Informing a GamePiece it is currently being drawn in the Mouseover Stack Viewer

Occasionally pieces will want to "draw themselves differently" inside of a Mouseover Stack Viewer. For this purpose they can check the property `DrawingMouseover` in e.g. a Calculated Property or a Label's "Follows Expression Value" field. Alternatively the property `DrawingMouseoverIndex` will contain 2 while drawing the mouseover, 1 otherwise; this can be useful as an index for a Follows Expression layer because it avoids the substantial performance cost of a full Beanshell expression e.g. `{ DrawingMouseover ? 2 : 1 }`.

Multiple Stack Viewers

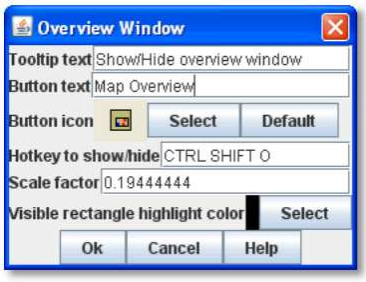
A Map Window can have any number of Stack Viewers, each with its own settings. You can use different Stack Viewers to view pieces of different types, on different Game Piece Layers, or with different attributes, and display them in different ways.

For example, a player's Map Window contains a stack of game pieces, as well as a stack of game money. To prevent them being stacked together, each of these piece types is assigned to a different Game Piece Layer. In addition, the money pieces each include a Marker Trait, *Value*, containing the value of the given piece.

- One viewer is set to display the game pieces, and has **Draw Pieces** enabled, with each piece's Basic Name displayed in a small label below. For **Include Individual Pieces**, *from layers other than those listed* is selected, and *Money* is entered. This viewer will now show any stack not on the Money layer, and display all the pieces in the stack.
- The second viewer has **Draw Pieces** disabled. In **Summary Text Above Pieces**, the setting *$sum(Value)$* is entered. For **Include Individual Pieces**, *from listed layers* is selected, and *Money* is entered. Now, when mousing over a stack of money, the total value of the money stack, but not the money pieces themselves, will be displayed.

Overview Window

The Overview Window adds a separate window that will be displayed whenever the main Map Window is displayed. The additional window will contain a view of the entire playing area at a smaller scale than displayed in the main Map Window. The area of the map currently visible in the Map Window is highlighted in the overview map with a colored rectangle. A player may click on the Overview window to center the Map Window at the point clicked on.



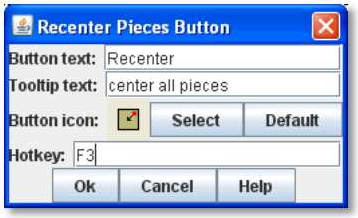
The scale of the overview window relative to the Map Window can be specified in the Scale Factor Property. You may also specify the color of the rectangle indicating the area visible in the main Map

Window.

The option has these settings:

- **Tooltip Text:** Tooltip shown when the cursor hovers over the button.
- **Button Text:** Overview window button text.
- **Button Icon:** Overview window button icon.
- **Hotkey to Show/Hide:** Keyboard shortcut to toggle Overview window.
- **Scale Factor:** Size of the Overview window compared to the current map view. For example, if the Scale Factor is 0.2, then the Overview window will show the full-scale map image at 20% size.
- **Visible Rectangle Highlight Color:** Color of the rectangle shown around the overview.

Re-center Pieces Button



A Re-Center Pieces button adds a button to the Map Window Toolbar button, appearing on the Main Controls toolbar, which will shift the position of all pieces on the map such that they are centered on the middle of the map as much as possible. This is useful for games where there are no absolute terrain features, such as many air, naval, and space combat games.

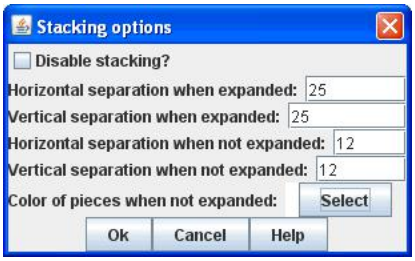
The option has these settings:

- **Button Text:** Text label for the button.
- **Tooltip Text:** Tooltip text displayed on mouseover.
- **Button Icon:** Icon used for the button.
- **Hotkey:** Keyboard shortcut for the button.

Because the size and layout of grids may vary widely, the Re-Center Pieces button may not place pieces exactly in the center of some grids, and some manual adjustment by players may be needed after re-centering.

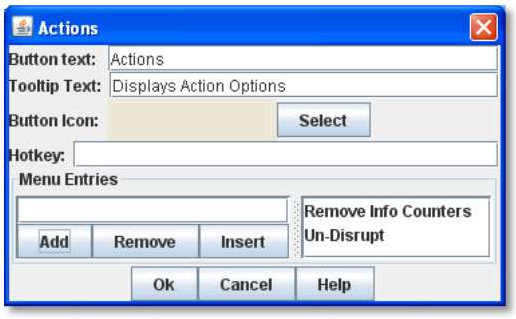
Stacking Options

Stacking Options determine how stacking is handled in this Map Window. The option may not be deleted.



- **Disable Stacking:** If selected, then pieces will never form stacks in this window.
- **Horizontal Separation When Expanded:** The distance in pixels from the left edge (right edge if negative) of a Game Piece in a stack to the edge of the piece above it when the stack is expanded.
- **Vertical Separation When Expanded:** The distance in pixels from the bottom edge (top edge if negative) of a Game Piece in a stack to the edge of the piece above it when the stack is expanded.
- **Horizontal Separation When Not Expanded:** The distance in pixels from the left edge (right edge if negative) of a Game Piece in a stack to the edge of the piece above it when the stack is compact.
- **Vertical Separation When Not Expanded:** The distance in pixels from the bottom edge (top edge if negative) of a Game Piece in a stack to the edge of the piece above it when the stack is compact.
- **Color Of Pieces When Not Expanded:** If set, then pieces below the top piece in a compact stack will be drawn as plain squares of this color and a black border. If not set (click **Select** and cancel the color-selection dialog) then pieces will be drawn normally.

Text Capture Tool



The Text Capture Tool adds a button to the Map Window Toolbar. Clicking the button will write a plain text summary of the contents of the map to a file, using the names assigned to the counters and the appropriate numbering of the board's Grid.

The option has these settings:

- **Button Text:** Text label for the Text Capture button.
- **Tooltip Text:** Tooltip text displayed on mouseover.
- **Button Icon:** Icon used for the Text Capture button.
- **Hotkey:** Keyboard shortcut for the button.

Toolbar Menu

The Toolbar Menu component enables you to group buttons from the Toolbar of the Main Controls window or a Map window into a drop-down menu on the Toolbar. Each button named in this component will be removed from the Toolbar and instead appear as a menu item in the drop-down menu.

- **Button Text:** The text of the button to be added to the Toolbar. Clicking the button will reveal the drop-down menu.
- **Button Icon:** Icon for the Toolbar button.
- **Hotkey:** Keyboard shortcut for revealing the drop-down menu.
- **Menu Entries:** Enter the text of the buttons that you wish to move to the drop-down menu. The menu item will have the same text. If the button uses an icon, the menu item will also use it.

Zoom Capability

Zoom capability enables re-scaling of a Board. You can add up to 3 buttons, for Zoom In, Zoom Out, and Zoom Select.

Zoom levels are defined as decimal numbers, each corresponding to a percentage of the full-scale map. For example, a 1000-pixel wide map, viewed at a Zoom level of .25 (25%), would appear to be 250 pixels across.

You can define an initial Zoom level. By default, this is 1.0 (which corresponds to a magnification factor of 100%), but you can select a different value. Zoom is defined in additional Zoom levels, which by default are defined at .39 (39%), .625 (62.5%), 1.0, and 1.6 (160%). However, you may add new levels to the list, or remove the defaults.

- Clicking the **Zoom In** button moves the current Zoom factor up the list of Zoom levels, from the initial value to higher values, making the map larger.
- Clicking the **Zoom Out** button moves the current Zoom factor down the list of Zoom levels, from the initial value to lower values, making the map smaller.
- Clicking **Zoom Select** enables the user to simply select a Zoom level from the defined levels.

The option has these settings:

- **Preset Zoom Levels:** A set of preset Zoom levels is listed. Each is identified by its scaling factor. For example, a Zoom level of .625 will show the board at 62.5% actual size. (A 1000 pixel-wide board would appear as 625 pixels across.) You can add a new level by entering a scaling factor in the text box and clicking **Add**. To remove a pre-set level, select it from the list and click **Remove**. To set the initial Zoom level (the one players see at game start), select the desired level and click **Set Initial.** The initial level will be marked with an asterisk (*).
- **Zoom In/Out/Select Tooltip Text:** Tooltip text for the button.
- **Zoom In/Out/Select Button Text:** Text label for the Zoom button.
- **Zoom In/Out/Select Icon:** Icon used for the Zoom button.

- **Zoom In/Out/Select Hotkey:** Keyboard shortcut used for the Zoom button.

Since the Zoom In and Zoom Out button functions are both duplicated b the Zoom Select button, you may wish to omit these buttons. To omit a particular Zoom button from the Map Toolbar, leave the text label and tooltip for the button blank. Then, next to the Icon for the button you do not wish to include, click **Select**, and then click **Cancel**. The button will not be displayed.

For example, to exclude the **Zoom In** button, next to **Zoom In Icon**, click Select, and then click Cancel. No Zoom In button will be included.

Map Grids

Map Grids help regulate movement and piece location. You can add one of the following types of Grid to a board: Hex, Rectangular, Irregular, and Multi-zoned.

Use of a Map Grid is optional. Although VASSAL Map Grids can help keep piece placement and movement tidy, hex and rectangular Grids in VASSAL are really most useful at the tactical scale, where range between hexes or squares may a factor in gameplay, and a Line of Sight Thread is used to track distances. For other games, such as those at the strategic scale, the printed grid included in the map image is often all that is necessary.

If you choose add a map grid to a board, each board in the same map window must have its own Grid, and each board may only have one grid (exception: see *Multi-Zoned Grids*, below.)

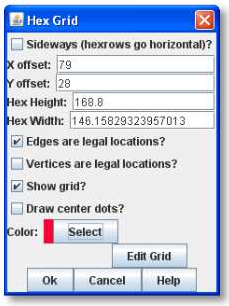
Like other components, map Grids can be copied and pasted from one Board to another.

By default, if a hex or rectangular Grid is imposed, pieces will *snap* to them, in which case all pieces will align neatly with the Grid cells. You can also enable snap for Irregular grids.

To turn off snap, choose cell edges or vertices as legal locations. (You can also have some pieces ignore snap by assigning them the Does Not Stack Trait. See page 46 for more information.)

Hex Grid

A Hex Grid is a standard hexagonal Grid for regulating movement on a Board. This type of Grid has these options:



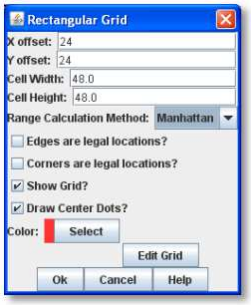
- **Sideways:** Check this box to make the hex rows of the Grid run right-to-left instead of top-to-bottom. (Setting the Grid to be Sideways switches the meanings of horizontal/vertical and x/y below.)

- **X,Y offset:** The horizontal and vertical position of the center of the first hex of the Grid.
- **Hex Height/Width:** In pixels from hex center to hex center. If you specify only the height, the width will adjust, or you can create oblong hexes by also specifying a width
- **Edges/Vertices are Legal Locations:** If selected, pieces can be placed on cell edges or corners, instead of only at hex centers.
- **Show Grid:** If selected, then the Grid will be drawn over the Board image using the specified color.
- **Draw Center Dots:** If selected, a dot will be drawn at the center of each hex in the specified color. You can add numbering to this type of Grid; see Grid Numbering on page 34.

Rectangular Grid

A standard rectangular Grid for regulating movement on a Board. This type of Grid has these options:

- **X,Y offset:** The horizontal and vertical position of the center of the first cell of the Grid.
- **Hex Height/Width:** in pixels of a single cell.
- **Edges/Corners are Legal Locations:** If selected, pieces can be placed on cell edges or corners, instead of only at cell centers.
- **Show Grid:** If selected, then the Grid will be drawn over the Board image using the specified color.
- **Draw Center Dots:** If selected, a dot will be drawn at the center of each cell in the specified color.



You can add numbering to this type of Grid; see Grid Numbering on page 34.

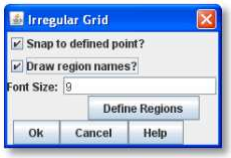
Irregular Grid

An irregular Grid is used for area-based games. It enables you to define a set of named Regions at arbitrary locations. These named Regions will act like the cell center points on hex or rectangular Grids. Pieces can be made to snap to the nearest named point, and their location will be reported as the nearest named point.

For maps with very irregularly shaped areas, you may need to specify more than one Region point in each area, each with the same name.

This type of Grid has these options:

- **Snap to Defined Point:** If selected, a Game Piece moved on the board will snap to the nearest defined Grid point.
- **Draw Region names:** If selected, the names of the Regions will be drawn on the map.
- **Font Size:** The font size used to draw the names.
- **Define Regions:** Click to display a window for defining the Regions. To add a new Region, right-click anywhere on the board and pick **Add Region**. To remove a Region, right-click on an existing Region's name and pick **Delete Region.** To change a Region's name or relocate it, click **Properties**, and then enter the new values.



Multi-Zoned Grid

A multi-zoned Grid enables you to define any number of areas on a board. Each area, called a Zone, can have its own Grid type and naming format, which takes precedence over the default Grid. For example, a board with a hex Grid may have zones along the edge for a turn track or force pools. Pieces will snap to positions in the appropriate Zone and auto-reporting will use text supplied by the zone.



Use of a multi-Zoned Grid is not recommended for a map with many Zones.

This type of Grid has these options:

- **Zone:** Each zone can have an arbitrary shape, which you specify in the Define Shape dialog. Each zone may define its own Grid. When defining a zone's Grid, the offsets and numbering are relative to the edge of the overall board, not the zone's edge.
- **Name:** The name of the Zone.
- **Location Format:** A Message Format that will be used to define the location of a point for auto-reporting of moves: name is the name of this Zone, GridLocation is the location name according to this zone's Grid.
- **Define Shape:** Hit this button to bring up a dialog for defining the shape of this zone. To create the initial shape, drag the mouse to define a rectangle. Then right-click to add new points and use the mouse to drag points to their final locations. Delete a point by clicking on it and pressing the Delete key.
- **Use Board's Grid:** If selected, then this Zone will use the Grid from the containing board instead of defining its own Grid.
- **Use Highlighting:** If selected, you must also specify the name of a Property. The value of the

Property will determine which Zone Highlighter is used to draw the zone.

- **Zone Highlighter:** Any number of Zone Highlighters can be added to a Multi-Zone Grid. Any Zone whose highlighting Property matches the name of a Zone Highlighter will be drawn with that highlighter, which overlays a colored pattern over the shape of the Zone.
- **Name:** The name of the highlighter.
- **Color:** The color of the highlight.
- **Coverage:** Select Entire Zone to overlay the entire shape of the zone. Select Zone Border to overlay only the border of the Zone.
- **Style:** Select from solid color, striped diagonal lines, crosshatched diagonal lines, or an image that you specify.
- **Opacity:** Select the transparency of the overlaid color or image.



If a given point does not fall within any of the defines Zones for a Multi- zone Grid, the default Grid is used. The default Grid may be any of the usual types of Grid: hex, rectangular or irregular.

Zone Highlighters

Any number of Zone Highlighters can be added to a Multi-Zone Grid. Any Zone whose highlighting property matches the name of a Zone Highlighter will be drawn with that highlighter, which overlays a colored pattern over the shape of the Zone.

- **Name:** The name of the highlighter.
- **Color:** The color of the highlight.
- **Coverage:** Select Entire Zone to overlay the entire shape of the zone. Select Zone Border to overlay only the border of the Zone.
- **Style:** Select from solid color, striped diagonal lines, crosshatched diagonal lines, or an image that you specify.
- **Opacity:** Select the transparency of the overlaid color or image.

Zone Properties

A Zone may contain Global Properties. Zone Properties may not have a Change-Property Toolbar button, but can be modified by a Set Global Property Game Piece Trait.

To assign a Global Property to a Zone,

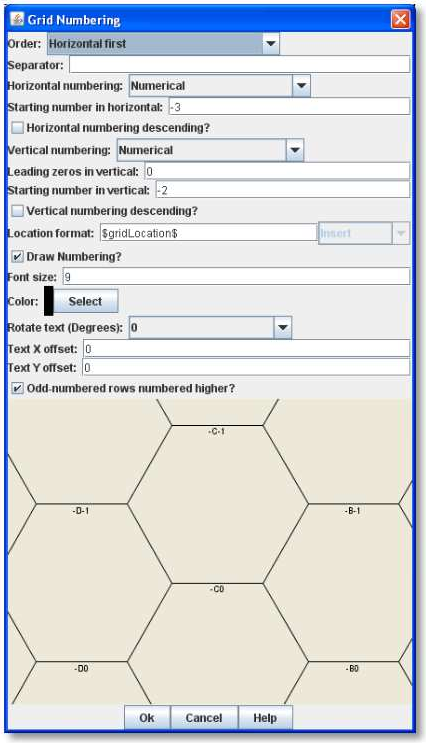
1. Right-click the Zone and pick **Add Global Property**.
2. In the **Global Property** dialog, enter name and other settings for the Property.
3. Click **Ok**.

For more about Global Properties, see page 88.

Adding Different Grid Settings to a Board

Multiple Grids can be added to a Board using Zones. Grids are added at the Board level, not the Map level, and so need to be set on each Board that makes up your map. Follow this procedure for each Board:

1. First create a board with a Multi-zoned Grid.
2. Create a standard Hex, Rectangular or Irregular Grid that covers most of the board. This is the 'default' or 'background' Grid that will be used for all areas of the Map not covered by a Zone.
3. For each area of the Board that is to have a different Grid, create a Zone. Don't click the **Use Board's Grid** button, as this will force the Zone to use the Grid you specified in step 2.
4. Right-click on the newly created Zone and you can now add a Hex, Rectangular or Irregular Grid that will apply only within that Zone.
5. If Zones overlap at a given point, the Zone defined first in the module (that is, topmost in the Module Editor) will take precedence at that point.



Grid Numbering

You can add Grid numbering to any hex or rectangular Grid. (Numbering is not applicable to the other Grid types.)

- **Order:** Label cells by row/column vs. column/row

- **Separator:** Text to place between the row and column, such as a comma
- **Numbering:** Alphabetical (A, B, C, … AA, BB, CC, etc.) vs. numerical (1,2,3…)
- **Descending:** If selected, numbering of rows and columns begins on the bottom right edge of the board.
- **Leading Zeros: Number of leading zeroes in each row or column**

number. One leading zero means to always use two digits for the row/column. Two leading zeros mean always use three digits, and so on.

- **Starting Number:** The number of the first cell ('A' == 0 if using alphabetic numbering).
- **Location Format:** The Message Format for reporting locations within a Map Window (for example, for move reporting): GridLocation is the name as drawn on the sample Grid. This is useful for pre-pending a board name, for example.
- **Draw Numbering:** If selected, the numbering of the Grid will be drawn on top of the board image.
- **Font size:** Size of the font to use when drawing the numbering.
- **Color:** Color to use when drawing the numbering.
- **Rotate Text:** Orientation of the numbering text.
- **Text X Offset:** Distance in pixels to the right (relative to the text's orientation) of its default position that the text will be drawn. By default, text is center-justified at the top of the cell.
- **Text Y Offset:** Distance in pixels downward (relative to the text's orientation) of its default position that the text will be drawn. By default, text is center-justified at the top of the cell.
- **Odd-Numbered Rows Numbered Higher:** For hex Grids only. If selected, then the first number of staggered columns on the Grid will be one greater than non-staggered columns.

Adding a Grid to a Board

To add a Grid to a board,

1. Select the **[Map Window]** node that contains the board.
2. Select the **[Board]** node.
3. Right-click the node and pick the type of Grid you would like to add from the list of commands.
4. In the dialog, configure the Grid as desired.
5. Click **Ok**.

To add Grid numbering to a hex or rectangular Grid,

1. Select the **[Board]** node that contains the hex or rectangular Grid.
2. Right-click the node and pick **Add Grid Numbering**.
3. In the **Grid Numbering** dialog, configure the Grid numbering as desired.
4. Click **Ok**.

Aligning a Map Grid with a Printed Grid

Some game board images already include a printed hexagonal or rectangular Grid. Of course, your module Grid should align with the printed Grid as closely as possible. The Module Editor has a number of tools to help you align a Grid.

*For better appearance, make a Map Grid invisible (**Draw Grid** is de-selected) if the Grid is already drawn on the printed map image.*

To align a hex or rectangular Grid with a printed Grid,

1. In the Module Editor, right-click the **[Hex Grid]** or **[Rectangular Grid]** node you wish to edit, and select **Properties**.
2. On the dialog, select **Draw Grid** and **Draw Center Dots.**
3. In **Color**, select a highly visible color.
4. Click **Edit Grid.**
5. On the **Edit Grid** dialog,
    - Use your arrow keys to shift the offset of the Grid. (Hold Shift down to increase the speed of the Grid movement.)
    - To resize the cells, use these keys: Ctrl-Down Arrow to increase the vertical cell dimensions. Ctrl-Up Arrow to decrease the vertical cell dimension. Ctrl-Right Arrow to increase the horizontal cell dimension. Ctrl-Left Arrow to decrease the horizontal cell dimension.
6. When the Grid aligns with the printed Grid, click **Save**.
7. Deselect the **Draw Grid** and **Draw Center Dots** checkboxes, so the VASSAL-imposed Grid is invisible.
8. Click **Ok**.

Guidelines for Grid Alignment

Aligning a Grid component with a printed map Grid can be tricky, particularly for hexagonal Grids. Follow these guidelines to help ensure an accurate Grid placement.

- Make the Grid and center dots a highly visible color when working on a Grid. (You can turn off the **Draw Grid** setting later, when you finalize the board.)
- Try to align the grid in the upper left-hand corner of the map. Then, move to the lower right-hand portion of the map. Align this, and then re-check the upper left-hand corner again. This will show you how much you might have to deviate from a perfect alignment to have pieces generally centered throughout the map, if both corners do not align exactly.
- Work on one axis a time:

o. Adjust the cell height first. Change the cell height slowly with the Ctrl-Up/Down Arrow keys until the Grid hexes are approximately the same height as the map hexes. Then, using the Up/Down Arrow keys, adjust the vertical offset to align them better. Fine-tune the cell height and cell placement.

o. Now, leaving cell height unchanged, work on cell width in the same way, using the Ctrl-Left/Right Arrow and Left/Right Arrow keys. Fine-tune the cell width and cell placement. Adjust the Hex width until you get a repeating pattern showing the hexes are about the same size.

The key for successful alignment to always adjust the cell height and vertical offset first, and get that right before working with the width and horizontal offset.

Sides

Sides represent different players in the game. Defining Sides in a module is optional.

- If you define no module Sides, then all windows and all Game Pieces are visible and accessible to all players.
- However, if you wish to create components that are accessible only to one player in a game (such as Private Windows), then you *must* define player Sides. Sides are used for Private Windows, Player Hands, and the Game Piece Traits Mask, Invisible, and Restricted Access. If these components or Traits are not used in your game, then defining Sides may not be necessary for your game.

Organizing Sides

You can create these types of Sides. (Note that these types are not specified in a module by name, but are a way of organizing Sides conceptually, for the players.)

- **Single:** With a single Side, each player represents a single group in the game. For example, in a World War II game, the German Side can control German units, but no others.
- **Compound:** With a compound Side, each player may represent more than one group in the game. Compound Sides could be mixed with single Sides for large multiplayer games that might sometimes be played by few players. For example, in a World War II game, the *Axis* Side can control both German and Japanese units, and the *Allies* would take the remaining forces. These two Sides are intended for a game where there are only two players. If the same game were played by 4 players, the players would be expected to select either the German, Japanese, American, Russian and British Sides, so in addition to *Axis* Side and *Allies* Side, there are also *German, Japanese, USA, Russia* and *UK* as Sides.
- **Solitaire:** A Solitaire Side can access all pieces and boards, to make it easier for solo players to play the game. Since a Solitaire Side can access any game components, there is no need to click **Retire** at the end of each turn to switch Sides and grant access. Inclusion of a Solitaire Side can be a useful way to play games that are played both solo and multiplayer.
- **Referee:** A Referee or Master Side is created like a Solitaire Side, but has access to anything that can be accessed by other players. In addition, a referee may have Private Windows or Game Pieces that only the referee can use.

Retiring from a Side

A Player can relinquish a current Side by clicking the **Retire** button on the Main Controls Toolbar. The button is configurable with these settings:

- **Button Text:** Text of the **Retire** button.
- **Button Tooltip:** Tooltip shown on mouseover for the **Retire** button.
- **Button Icon:** Icon used for the **Retire** button.

Adding Sides to the Module

Sides can be named anything you choose, and should reflect the groups or forces in the physical

game. They can be named for the color of the units, their nationality, or be simply numbered Player 1, Player 2, and so on. Each Side must have a unique name.

A module may have any number of Sides defined.

All module Sides will appear in the drop-down list offered to players at game start, even if some of them are not used in a particular scenario. There is no way to suppress the display of some Sides from this drop-down.

In addition, each side must have a unique password, which is chosen by the player who plays the Side at game time. The privacy of a Side is ensured by player passwords. Only one player may join a Side. If one Side is taken, when joining a game, players will be prompted to take one of the remaining available Sides.

See the *VASSAL User's Guide* for more information about Sides and passwords.

To add to the list of available Sides,

1. In the Configuration Window, right-click the **[Definition of Player Sides]** node, and choose **Properties**.
2. In the **Definition of Player Sides** dialog, under **Sides Available to Players**, enter the name of a Side and click **Add**. The Side is added to the list.
3. Repeat Step 2 for each additional Side.
4. Specify text, tooltip and icon for the **Retire** button.
5. Click **Ok**.

When the game begins, the Sides are presented for player selection in the order you specify.

Observer Side

The *Observer* Side is included by default in all modules, and may not be deleted. As the name implies, the Observer can watch a game, but will not have access to any Side-restricted components. A game may have any number of Observers logged in at the same time.

The Observer Side is also important when creating Pre-Defined Setups. See page 97 for more information.

Next Steps: Restricting Components By Side

The process of creating Sides merely makes a list of available Sides for the players to choose from. To restrict components to one or more of these sides, you must now configure Side access for individual components, such as with Private Windows or Player Hands (see *Maps and Boards* on page 20) or using the Restricted Access, Mask, or Invisible Traits on Game Pieces (see *Trait Descriptions* beginning on page 42).

Sides assigned to restricted components must match one or more existing Sides exactly as it is spelled in the List of Available Sides. For example, if you restrict pieces in the World War II game to

the “UK” Side, then listing “U.K.” as a Side for each component will not correctly grant access. Side names are also case-sensitive.

Game Pieces

Game Pieces are what you move around on the map in order to play the game. In some games, pieces are known as *units*, *counters*, or *tokens*. Pieces can include ordinary Game Pieces, cards, game money, movement tokens, markers, and even map tiles.

Game Piece Palette

A Game Piece Palette is a tool for generating and organizing Game Pieces. During the game, players draw pieces from one or more palettes and place them on the map. There is no limit to the number of pieces that can be generated by a Game Piece Palette. Pieces will appear in the palette in the order they are listed in the Configuration Window.



Each Game Piece Palette has these attributes:

- **Name:** If the palette appears in its own window, this will be used for the title.
- **Hidden:** If selected, then this Game Piece Palette will not appear at all during play. This is useful if you need to define pieces for a default Setup but don't want to allow players to create new pieces during play. You must restart VASSAL for this change to take effect in the Module Editor.
- **Button Text:** The text on the Toolbar button that shows and hides the Game Piece Palette.
- **Button Icon:** The icon image on the Toolbar button.
- **Hotkey to Show/Hide:** A keyboard shortcut for the Toolbar button. Toggles visibility of the Palette.

> An infinite number of Pieces can be drawn from a Game Piece Palette. However, some games have a limited number of pieces available on purpose. Too many pieces may actually affect the game □s balance and playability. In such games, your module should have the same limitations as the actual game. If the quantity of a Game Piece is limited in a game, it□s probably better to use At-Start Stacks than a Game Piece Palette. See page 24 for more information.

Palette Sub-Components

A Game Piece Palette is highly configurable, and can contain any combination of tabs, lists, and pull-down menus containing individual Game Pieces. For example, a Scrollable List could include a Panel, which includes individual Game Pieces.

- **Tabbed Panel:** A panel with tabs, each of which corresponds to a Panel or other Tabbed Panel

subcomponent. The label of the tab will be the name of the subcomponent.

- **Panel:** A panel that can contain Single Pieces, Tabbed Panes, or other panels. Select **Fixed Cell Size** to specify a fixed number of columns for the panel. Otherwise, the sub-components will appear in a single row, or a single column if the **Vertical layout box** is checked.
- **Pull-down Menu:** A pull-down menu in which each menu item corresponds to a subcomponent. The name of the menu item will be the name of the subcomponent.
- **Scrollable List:** A scroll list in which each entry corresponds to a subcomponent. The name of the entry will be the name of the subcomponent.
- **Single Piece:** A Game Piece that can be dragged onto a playing area. (Most Single Piece sub-components will be part of another sub-component, grouped with similar pieces.)

By default, a new module includes an empty Game Piece Palette, but you can modify the default, or create as many new ones as you need.

Docked Palette

By default, the first palette in the Editor (that is, at the top of the list in the Editor window) will be displayed docked (attached) to the module main controls and Chat Window. All other palettes will be undocked (detached).

If a player has the **Use Combined Application Window** preference checked, then the first (topmost) Game Piece Palette in the Configuration Window will dock into the main controls window to the left of the chat area. All other Palettes will appear in their own window.

To detach all palettes from the toolbar for all players, use an empty hidden palette as the first one in the list.

Game Pieces: Creating Game Pieces

Creating a Game Piece Palette

To create a Game Piece Palette,

1. Right-click the **[Module]** node and select **Add Game Piece Palette.**
2. In the **Game Piece Palette** dialog, specify the settings for the palette.
3. Click **Ok**.
4. Right-click the new **[Game Piece Palette]** node and pick a sub-component to add.
5. Continue adding subcomponents as needed.

After a palette is created, you can then create individual pieces.

Pre-Setting Pieces in a Game Piece Palette

Pieces in a Game Piece Palette can be pre-set in particular states based on the Traits they possess. Whenever the pieces are drawn from the palette, they will be in the pre-set state.

For example, you would like a rotatable piece to start each game rotated 90 degrees to the right. In

the VASSAL Player, right-click the piece in the palette, select **Rotate** from the piece's Command Menu, and rotate the piece to desired position. Now save the module. The pieces will be displayed in the rotated state you saved, and new pieces dragged from the palette will be created rotated 90 degrees to the right.

In order to pre-set a Trait, the piece must possess the Trait directly and may not inherit it from a Prototype. (For pre-setting Traits in a Prototype, see page 67.)

Displaying Large Pieces in the Palette

Each level of the palette allows a scale factor to be set, controlling the size that pieces are drawn. Thus you can specify a smaller scale for palettes containing large pieces.

Creating Game Pieces

A Game Piece is defined by its *Traits*. Each Trait gives a Game Piece one or more kinds of specialized behavior or attributes.

You can assign any number of Traits to your piece, to reflect its usage in the game.

By default, each piece includes the Basic Piece Trait (see page 44), which defines the name of the piece and the basic image used. The Basic Piece Trait may not be deleted.

Trait Order

The order of Traits as they appear in the piece's **Properties** dialog is crucial when defining a Game Piece's behavior. Traits are evaluated from the bottom of the list to the top. *A Trait will only affect those Traits above it in the *Properties* dialog.*

For example, if you define a Rotate Trait on a Game Piece, and then a Text Label Trait below that, when the piece is rotated, the Text Label will not rotate.

If you were to move the Text Label Trait *above* the Rotate Trait, then when the piece is rotated, the Text Label would rotate along with the piece.

Trait order is often the first place to look when diagnosing a Game Piece that is not behaving the way you intended. Always check the Trait order to determine if your Traits are being applied in the intended sequence (that is, bottom of the list to the top).

Trait order is *crucial* when defining a Game Piece's behavior. A Trait will only affect those Traits *above* it in the **Properties** dialog.

You may get unexpected results if some Traits are not placed at the end of the list of Traits, where they can affect all other Traits. Examples of Traits that should usually be placed at the bottom of the Properties list include Mask, Invisible, and Restricted Access.

Using the Piece Properties Dialog

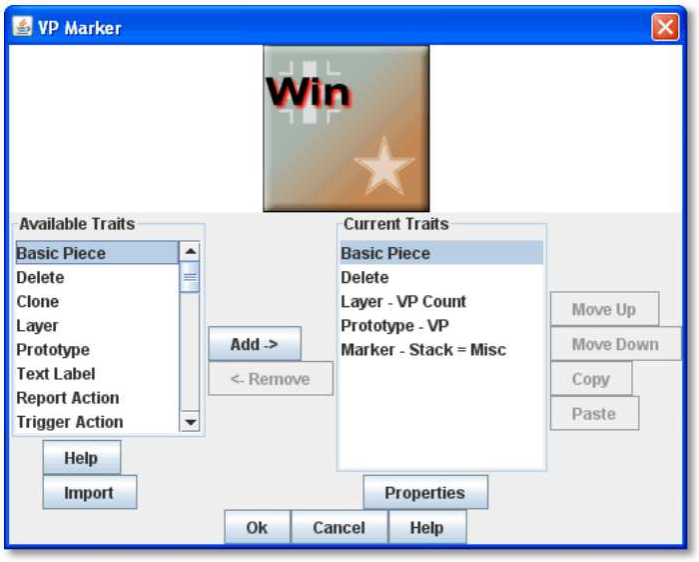
To create a Game Piece,

1. In the Module Editor, select (or create) a **[Game Piece Palette]** node.
2. Select where in the palette you want the piece to appear (in a scrollable list, a tab, etc.)
3. Right-click and choose **Add Single Piece.**
4. Under **Current Traits**, Basic Piece is selected. Click **Properties**.
5. In the **Basic Piece Properties** dialog, do one of the following:
    - Double-click the left side of the dialog, and then browse to the location of the image file you want to use for the piece, or,
    - Using the drop-down list, select an existing image file from the list of files that are already in the module.
6. In **Name**, enter the basic name of the piece.
7. Click **Ok.** The piece is added to the palette.

*You can copy and paste Traits between pieces (and Prototype Definitions). In the **Properties** dialog of the first piece (or Prototype), select the Trait you want to copy, and click **Copy**. Open the **Properties** dialog of the second piece (or Prototype), and click **Paste**. The Trait is copied to the where you pasted it.*

To add Traits to a Game Piece,

1. Select the Game Piece in its palette and click **Properties**.
2. On the **Properties** dialog, select a Trait from the **Available Traits** list, and click **Add**. The Trait is moved to the **Current Traits** list.
3. Select the Trait, and then click **Properties.**
4. In the dialog, specify the settings for the Trait.
5. Repeat steps 2-4 until the behavior of the piece has been specified.
6. Click **Ok**.



To change the placement order of an assigned Trait,

1. In the **Current Traits** list, select the Trait to move.

2. Click **Up** or **Down** to move the Trait up or down in the list.
3. When complete, click **Ok**.

Traits and the Command Menu

A. Game Piece's right-click Command Menu will display Trait-related commands in reverse order from the way they appear in the Trait list. For example, if the Clone Trait were the *last* Trait defined on a Game Piece (that is, bottom-most on the Trait list), then the corresponding **Clone** command would be the *first* one displayed in the piece's Command Menu.
    - If no text label is specified for a command, then the command will not be displayed in the menu. However, the command's keyboard shortcut could still be used as if the menu item were visible. (Commands that are solely part of a Trigger Action often omit the text label, so the commands will not appear on the menu outside of the Trigger command.)
    - You may not omit the Keyboard Shortcut for a Trait (if the Trait Properties dialog prompts for one).

For example, you define a Delete Trait on a Game Piece with a keyboard shortcut of Ctrl-X. You leave the value of Command Name for the Trait blank. As a result, no Delete command shows up in the Command Menu. However, the keyboard shortcut Ctrl-X could still be used in Global Key Commands or Trigger Actions.

Keyboard Shortcuts (Hotkeys)

Pressing a command's keyboard shortcut (hotkey) when the piece is selected will invoke the corresponding command, just as if the menu item was selected. For example, a Move Fixed Distance Trait could be defined to use the Ctrl-M shortcut. A player would hold down Ctrl and M simultaneously, with the piece selected, to launch the command.

Hotkeys can also be invoked by automated commands. For example, Trigger Action Traits make use of hotkeys when referring to a sequence of commands. In every respect, a Hotkey invoked by automated commands will work the same as if a player had pressed the key combination on a keyboard.

You can define any unique keyboard shortcut you want as a Hotkey for a particular command. To make it harder to press them accidentally, keyboard shortcuts are usually comprised of more than one key, such as Ctrl-X or Alt-Shift-K.

A keyboard shortcut could be composed of any number of keys, but generally use 2 or 3 keys; usually a letter or number combined with one of the following keys: Ctrl, Alt/Option, Shift, or Meta/Command.

To make them more memorable for players, when assigning keyboard shortcuts, use key combinations that are reminiscent of the command itself. (For example, Ctrl-D would be an easily remembered shortcut for a Delete command.)

Use these guidelines when assigning keyboard shortcuts.

- Keyboard shortcuts should be unique for a given type of piece. If not, when the shortcut is

invoked, more than a single command could be fired at once, with possibly unexpected results.

- Avoid using keyboard shortcuts that players could type inadvertently. For example, a single capital letter M would not be a suitable shortcut, nor would Shift-M, because players could easily type either in the Chat window during ordinary conversation. However, Ctrl-M or Ctrl-Shift-M would both be suitable.
- Be careful about assigning hotkeys to keys that invoke special functions on your computer. Caps Lock, Backspace, Delete, Home, End, Enter/Return, and so on, are not appropriate for use as hotkeys. Similarly, the Function (F1-F9) keys at the top of a standard keyboard may serve as hotkeys for various Windows or MacOS functions, and pressing them could cause unexpected operating system functions to be invoked instead of the desired piece command.

Trait Descriptions

The following Traits are available for use with Game Pieces.

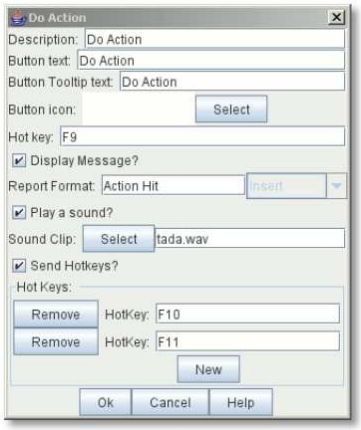
Action Button	Movement Trail	Area of Effect	Non-Rectangular
Basic Piece	Place Marker	Can Pivot	Play Sound
Can Rotate	Property Sheet	Clone	Prototype
Delete	Replace With Other	Does Not Stack	Report Action
Dynamic Property	Restrict Commands	Global Hotkey	Restrict Access
Global Key Command	Return to Deck	Invisible	Send to Location
Layer	Set Global Property	Mark When Moved	Spreadsheet
Marker	Sub-menu	Mask	Text Label

Action Button

Action Button places a virtual button on your piece. Clicking within the specified rectangular region on the piece will invoke an action just as if the corresponding key command had been typed.

An Action Button Trait has these attributes:

- **Invoke Key Command:** The keyboard command to be invoked.
- **Button X-offset:** The horizontal position of the upper-left corner of the rectangle, in pixels from the center of the piece. Negative numbers are toward the left.
- **Button Y-offset:** The vertical position of the upper-left corner of the rectangle, in pixels from the center of the piece. Negative numbers are toward the top.
- **Button Width:** The width in pixels of the button.
- **Button Height:** The height in pixels of the button.



This Trait does not alter the way a Game Piece is drawn, so the Basic Piece or a Layer should be used to supply a visual cue to the player that the button exists.

The Action Button Trait is never affected by the Can Rotate Trait (no matter where the Action Button is placed in the Trait order).

To make a button that can be activated and deactivated, combine an Action Button with a Layer and a Trigger Action.

Example: A Game Piece representing a spaceship has a self- destruct action that can only be activated when the energy reaches the minimum level. A Layer named Energy is used to represent the energy level. The image for the lowest level of the layer adds an icon for a self-destruct button. An Action Button Trait uses the boundaries of the button icon and invokes Ctrl-ALT+T. A Trigger Action watches for Ctrl- ALT+T and invokes the keyboard command for self-destruct when the Properties match Energy_Level = 1.

NOTE: The Action Button Trait is not related to the Action Button module component (see page 84).

Aligning an Action Button

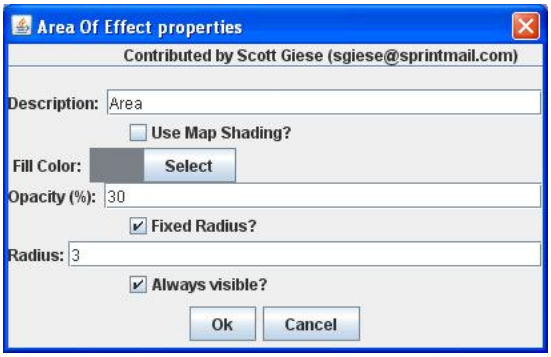
Use this formula for aligning an Action Button:

- X offset = [width of piece / 2] – (X Position). Then change sign, from positive to negative, or vice versa.
- Y offset = [height of piece / 2] – (Y Position). Then change sign, from positive to negative, or vice versa.

For example, a 100x100 pixel game piece, with button position on the piece at 25x60 pixels, would have these X and Y offset values:

- X-offset = 100 / 2 = 50 - 25 = 25. Change sign, so final X-offset value is -25.
- Y offset = 100 / 2 = 50 - 60 = -10. Change sign , so final Y-offset value is 10.

Area of Effect



The Area of Effect Trait enables you to graphically highlight an area surrounding a Game Piece. The area is shaded with a specified color and transparency. Alternatively, you can point to a Map Shading component, contributing to the area that it draws.

Area of Effect has these attributes:

- **Use Map Shading:** If selected, then the area of this Trait will be added to the area drawn by the named Map Shading component (or subtracted from that area if it is of type Background). If not selected, then each piece with this Trait will draw its own area, with overlapping areas shaded darker.
- **Fill Color:** The color of the Area of Effect.
- **Opacity:** The opacity of the Area. 100% is completely opaque. 0% is completely invisible.
- **Radius:** Distance, in local Grid units, from the Game Piece that will be highlighted. If the piece is on a board with a Rectangular Grid or Hex Grid, this distance is in Grid units and the shaded area will conform to the Grid. Otherwise, it will be a circle with the given radius in pixels.
- **Always Visible:** If selected, the area is always highlighted when the piece is drawn on a Map.
- **Toggle Visible Command:** If not always visible, this is the Command Menu item to show/hide the highlighted area.
- **Toggle Visible Keyboard Shortcut:** If not always visible, the keyboard shortcut to show/hide the highlighted area.

Basic Piece

All Game Pieces have the Basic Piece Trait. The Basic Piece Trait consists of a Game Piece name and assigned image.

Game Piece Name

A game piece name can be any alphanumeric string of text. The name can include space characters ( ).



Game Piece Image

Your image can come from one of these sources:

- You can import an externally created image from outside

VASSAL (such as a scanned piece image, or image you have otherwise created).

- You can create an image using the Game Piece Image Definition component. See page 69 for more information on Game Piece Image Definitions.
- You can use an image that already exists in the module. The image selector drop-down, found in the **Basic Piece** dialog, includes an alphabetical list of every image in the module.

Alternatively, you can create a composite piece image using the Layer Trait. See page 50 for more information.

Basic Piece System Properties

The following system Properties are defined for the Basic Piece Trait (and therefore defined for all Game Pieces). Remember that Property names are case-sensitive.

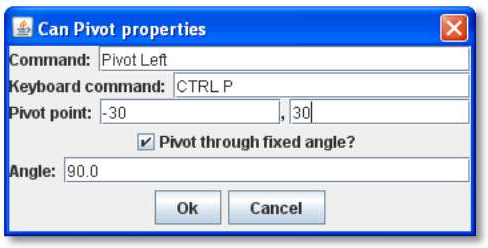
Property	**Description**
BasicName	Name of the Basic Piece Trait.
PieceName	Full name of the piece, including all Traits.
PlayerSide	Side of the current player.
LocationName	Name of the current location, as determined by the local Grid. If no Grid is assigned to the Board, the value will be □offboard.□
CurrentMap	Name of the current Map Window.
CurrentBoard	Name of the current Board.
CurrentZone	Name of the current Zone.
CurrentX	The current map X coordinate.
CurrentY	The current map Y coordinate.

DeckName	Name of the Deck, if the piece is currently stacked in one.
Selected	Boolean. Has a value of true when the piece has been selected with the mouse.
OldLocationName	Name of the previous location, as determined by the local Grid (after the piece has been moved by drag-and- drop movement).
OldMap	Name of the previous Map Window (after the piece has been moved by drag-and-drop movement).
OldBoard	Name of the previous Board (after the piece has been moved by drag-and-drop movement).
OldZone	Name of the previous Zone (after the piece has been moved by drag-and-drop movement).
OldX	Previous map X coordinate (after the piece has been moved by drag-and-drop movement).
OldY	Previous map Y coordinate (after the piece has been moved by drag-and-drop movement).

Can Pivot

Can Pivot enables a Game Piece to pivot around a fixed point relative to its current position. A piece with Can Pivot must also include Can Rotate, which must appear before (below) the Can Pivot Trait. The Trait has these attributes:

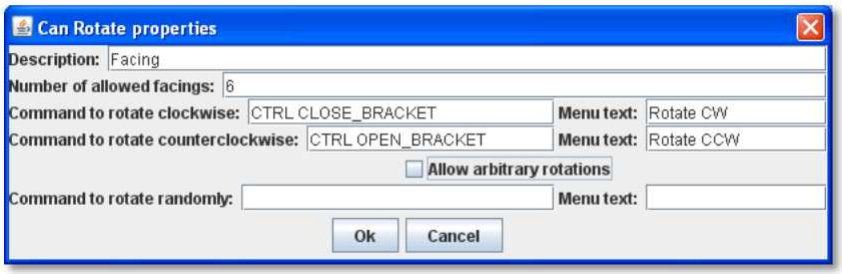
- **Command:** The Command Menu item to pivot the piece.
- **Key Command:** The keyboard shortcut of the command.
- **Pivot Point:** The location, relative to the center of the piece and its current facing, around which the piece will rotate. Positive numbers are down and to the right. *Example: For a Game Piece of size 40x40, a pivot point of 20,-20 will rotate the piece around its upper right corner.*
- **Pivot Through Fixed Angle:** If selected, then invoking the command will pivot the piece through the angle specified in the Angle field, in degrees clockwise. If left unselected, then invoking the command will allow the player to pivot the piece interactively by any angle by dragging the mouse.



Can Rotate

Can Rotate enables a Game Piece to be rotated through an arbitrary number of facings. The Trait has these attributes:

- **Description:** Description of the Can Rotate Trait.
- **Number of Allowed Facings:** You can choose the number of valid facings. For example, a hex-based game may have six possible facings, while a game with a square Grid game might have four (or eight, if corners are used). Each use of the command to rotate clockwise or counter-clockwise will rotate the piece one facing.
- **Command to Rotate Clockwise:** If specified, the keyboard shortcut to rotate clockwise, and the accompanying menu text.
- **Command to Rotate Counter-clockwise:** If specified, the keyboard shortcut to rotate counter-clockwise, and the accompanying menu text.
- **Allow Arbitrary Rotations:** If selected, then the user can drag the Game Piece to rotate it to any facing.
- **Command to Rotate Randomly:** If specified, this command will rotate the piece to a random facing (in one of the valid facings, if applicable).



Like other Traits, Can Rotate will affect only

those Traits that appear above it in the list of Traits

for a Game Piece. Traits below the Can Rotate Trait will be drawn on top of the rotated image.

Since the rotations are created on the fly from a bitmapped image, the image quality of a rotated counter may be lower than the unrotated version. You may get better image quality for your rotations by creating separate images for each rotation in an external image editor and putting them into different levels of a Layer.

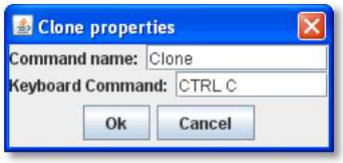
Can Rotate Trait System Properties

The Can Rotate Trait includes these system Properties. In the name of the Properties, <name> is the name specified in the attributes above.

Property Description

Property	Description

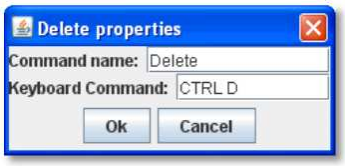
<name>_Facing	The current facing, if the number of facings is fixed.
<name>_Degrees	The current rotation angle, if arbitrary rotations are allowed.



Clone

Clone will duplicate the Game Piece during a game. The Trait has these attributes:

- **Command Name:** The Command Menu item to clone the piece.
- **Key Command:** The keyboard shortcut of the command.



Delete

Delete will delete the Game Piece from the game. The Trait has these attributes:

- **Command Name:** The Command Menu item to delete the piece.
- **Key Command:** The keyboard shortcut of the command.

Does Not Stack

A Game Piece with the Does Not Stack Trait will not form stacks with other pieces. In addition, a piece with this Trait can be assigned special treatment when it comes to selection and movement.

The Trait has these attributes:

- **Select Piece:** Controls how the piece is selected: either normally, never (can never be selected), only when the shift key is down (shift-click to select the piece), or only when the Alt and shift keys are down (alt-click to select the piece).
- **Move Piece:** Controls how the piece is moved: either normally, never (cannot be moved once placed) or only if selected (select piece, then click and drag to move).
- **Ignore Map Grid When Moving:** If selected, then this piece will not snap to the nearest Grid location.

Some uses for the Does Not Stack Trait include:

- In games that mix cards and counters, the Do Not Stack Trait can be assigned to cards, so that the cards can be placed on a map without interfering with stacks of counters. In addition, the cards will not form stacks and be generally easier to manipulate on screen.
- Pieces that represent map features, such as buildings, can use the *Move Piece - Never* option so that players do not inadvertently move them around.

Dynamic Property

A Dynamic Property Trait enables you to assign a custom Property to the Game Piece, and to define commands to change the value of the Property during play.

Setting a Property does not in itself give a Game Piece any particular behavior. The Property must be recognized by some other component in the module. Dynamic Properties are used by Global Key Commands and other components and often by custom Java classes.



The Trait has these attributes:

- **Name:** The name of the Property.
- **Value:** The value of the Property at the start of a new game.
- **Is Numeric?** If selected, then changes to the value of the Property will be restricted to integer values.
- **Minimum Value:** Numeric values will be restricted to no less than this number.
- **Maximum Value:** Numeric values will be restricted to no more than this number.
- **Wrap?** If selected, then when incrementing this numeric Property, values will wrap around from the maximum to the minimum.
- **Key Commands:** Adds any number of commands to the right-click drop-down menu for this Game Piece. Click the **New** button to add a new command and the **Remove** button to remove one. For each command, specify the text of the drop-down menu entry and the keyboard shortcut. The type defines how the Property value should change:

o. *Set value directly* sets the Property to a fixed value. You can set a numerical value or the value of another Property. (To specify a Property, enter the name of the Property in $-signs; for example, $ExampleProperty$.)

o. *Increment numeric value* adds a fixed value to the Property. You can set a number, or the value of another Property. (To specify a Property, enter the name of the Property in $-signs; for example, $ExampleProperty$.)

o. *Prompt user* displays a dialog for the user to type in a new value.

o. *Prompt user to select from list* displays a dialog with a drop-down menu for the user to select from.

Example: we define a Dynamic Property called Hit Points that represents the amount of damage taken by a warrior. Hit Points has a maximum level of 20, and a minimum of 0. We add a command to the Property with a Command Menu item of ***Change Value*** *and a shortcut of Ctrl-V. When the user selects* ***Change Value****, the module prompts for the new value of Hit Points. Dynamic Properties do not display their values on a Game Piece, but we could display the current value of each warrior's Hit Points using a Text Label or a Layer.*

If a Property's value always remains the same during the game, it may be better to define it using the Marker Trait instead.

See page 52 for more information.

Global Hotkey

The Global Hotkey Trait adds an action that invokes a Hotkey (that is, a keyboard shortcut for a Toolbar button) in the Main Controls windows or a Map Window. For example, you could use a Global Hotkey to trigger the firing of a Global Key Command Button or Dice Button.



Define the hotkey for the button you wish to invoke before creating the Global Hotkey Trait.

The Trait has these attributes:

- **Menu Command:** Command menu text.
- **Key Command:** Keystroke or Named Command of the menu item that initiates the command.
- **Global Hotkey:** The Keystroke or Named Command that will be applied to the Main Controls window.

EXAMPLE: A Dice Button component has been added to the Toolbar, and given the Hotkey F2. A Game Piece is given a Global Hotkey Trait with Menu Text **Roll Dice***, Keyboard Command* Ctrl-R*, and Global Hotkey* F2*. Now, selecting the piece and typing Ctrl-R or selecting* **Roll Dice** *from the Command Menu will roll the dice button just as if the player had clicked the button in the Toolbar or typed F2 from the keyboard.*

Global Key Command

The Global Key Command (GKC) Trait adds an action that applies a keyboard command to other pieces, similar to the Global Key Command component of a module or Map Window. A GKC Trait can potentially affect any pieces anywhere in the game, on any map.

The Trait has these attributes:

- **Description:** Description of the GKC Trait.
- **Menu Command:** Menu text of the command to activate the GKC.
- **Key Command:** Keystroke or Named Command of the menu item that initiates the GKC.
- **Global Key Command:** The Keystroke or Named Command that will be applied to other pieces.
- **Matching Properties:** The key command will only be applied to pieces with the specified Properties.
- **Restrict Range:** If selected, the command will only apply to pieces located within a specified distance of this piece.
- **Within a Deck, Apply To:** Select how this command applies to pieces that are contained within a Deck.

o. *No pieces* means that all pieces in a Deck ignore the command.

o. *All pieces* means that the command applies to the entire Deck.

  o. _ *Fixed number of pieces* enables you to specify the number of pieces (drawn from the top) that the command will apply to.

    - **Restrict Range:** Only others pieces within this distance, inclusive, of this piece will have the command applied to them. If the pieces are on a board with a Hex Grid or Rectangular Grid, then the distance is in units of the Grid. Otherwise, the distance is measured in screen pixels.

p. **Fixed Range:** If selected, then the range is specified as a fixed number. If unselected, then the range will be given by the value of the named Property.

  - **Suppress Individual Reports:** If selected, then any auto-reporting of the affected pieces will be disabled. Use the Report Action Trait to provide a summary message in their place.

Commands applied by Global Key Commands will be affected by piece ownership. If the GKC triggers a command that is restricted by side, the action may not take place as intended when the restricted side triggers the GKC (by button or other command).

EXAMPLE: A leader counter and infantry counters both have Marker Traits to specify their nationality and type. A Layer Trait represents the rallied state of an infantry counter, uses Ctrl A to activate the layer, and uses Rally as the name. A Global Key Command on the leader counter can select and rally all infantry counters within two hexes of the same nationality that are not rallied by specifying Range=2 and matching Properties type=Infantry && nation=$nation$ && Rally_Active=false.

Invisible

The Invisible Trait gives a Game Piece the capability to be made invisible (or, visible if the piece is already invisible). An invisible Game Piece will be seen as translucent by the hiding player but completely hidden from the view of the other players.

Use of the Invisible Trait will require you to define Sides in the game. See page 37 for more information.

The Trait has these attributes:



- **Key Command:** Keystroke or Named Command to toggle visibility.
- **Menu Command:** Menu text of the command to toggle visibility.
- **Background Color:** To the player who turned it invisible, the piece will appear transparent against a background of the specified color. To other players, it will not appear at all.
- **Can Be Hidden By:** Defines who may hide this piece (and see it once hidden).

o. *Any Player* means that any player may hide this piece, including observers.

o. *Any Side* means that any player who has been assigned a Side in a game (that is, not an observer) can hide this piece. If the player resigns and another player takes the Side, then the new player for that Side will be the owner.

o. *Any of the Specified Sides* enables you to enter a list of Sides. Only players assigned to one of the named Sides can hide the piece, but the players of all the listed Sides will be able to see and modify the piece. This is useful for referee players or games with multi-player teams.

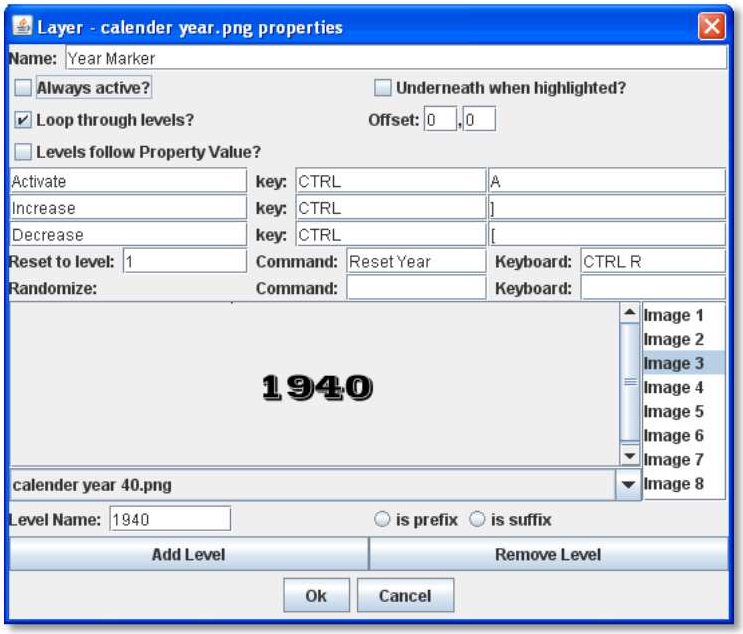
The Invisible Trait only hides those Traits that are above it in the list of Traits. In addition, movement Report Traits will not return any report on the movement of Invisible pieces.

Invisible Trait Properties

The Invisible Trait includes one System Property:

Property	Description
InvisibleToOthers	Has a value of true if the piece is hidden.

Layer



A Layer Trait is used for interactively changing the appearance of Game Pieces. Layers have a number of uses that include, but are not limited to:

- *Changing a piece's appearance:* A Layer Trait can be used to change a Game Piece's appearance, equivalent to flipping a two-sided counter to its reverse face. For example, a tank counter has two faces: one shows the tank at full strength and the other at depleted strength. The Basic Piece Trait could show the Tank at full strength and a Layer could show it at its depleted level. Where a physical counter may only have two sides, the Layer Trait can actually reflect any number of counter "faces".
- *Placing a status marker:* A Layer can substitute for placing a separate status marker on top of another piece. For example, in the actual board game, when a unit is targeted by other units, a separate counter is placed atop it that says "Targeted. In the module, a "Targeted" Layer can be created for units and a menu item added to toggle this marker on and off.
- *Creating a piece layout:* A Layer can be used to change the foreground or background images assigned to a Game Piece. For example, a Game Piece is defined with a blank image for the Basic Piece Trait. The background is defined as a Red or Blue Layer, and the foreground is defined as an Infantry symbol or Tank symbol. During the game, the same piece could be switched from red to blue background, and the symbol could be switched from Infantry to Tank, so one piece could actually be turned into 4 separate units.

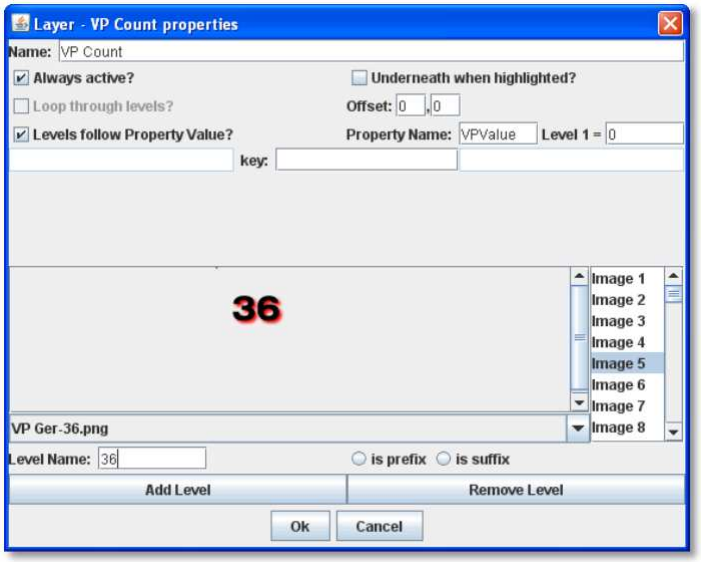
To simulate two -sided pieces where one face of the piece is hidden from one or more players, it's better to use the Mask Trait. See page 52 for more information.

Configuring a Layer

A Layer Trait consists of a number of 'levels", each of which has an image and a name. The Layer can be activated with a keyboard command, and players can change the current level during play. The image from the current level will be drawn whenever the Layer is activated. The Layer is drawn on top the Traits that appear above it in the list of Traits.

The Trait has these attributes:

- **Name:** The name of this Layer, used for reference during editing and as the prefix for the name of any Properties defined by this Layer.
- **Always Active:** If selected, then this layer is always active; that is, the current layer will always be displayed. If unchecked, then the layer must be activated (by the specified keyboard command) in order to display the current layer.
- **Underneath When Highlighted:** If selected, then this layer will be drawn underneath the rest of the piece when the counter has been highlighted (by clicking on it).
- **Loop Through Levels:** If selected, then increasing the level past the last one will loop through to the first level and vice versa. Otherwise, increasing the level has no effect if the current level is the last level.
- **Offset:** The images of a level are drawn with their center offset from the center of the underlying piece by a number of pixels specified by the offset boxes, with positive numbers giving an offset down and to the right. For example, if a layer image is 40x40 pixels and you want it to be drawn so that the lower-left corner is at the center of the Game Piece, set the offset to 20,-20.
- **Levels Follow Property Value:** If selected, then you can specify the name of a numeric Property that will determine the active level, rather than responding directly to keyboard events. A typical use will specify the name of a numeric Dynamic Property on the piece, or a Global Property. As the Property changes value, the level displayed will change as well. You can also specify the numeric value of the Property that should correspond to the first level of this Layer.
- **Activate/Increase/Decrease:** Specify the keyboard commands and Command Menu text that will activate the Layer and increase or decrease the current level. The Activate keyboard shortcut can specify a string of characters, such that the layer is activated only when all the corresponding keys have been pressed. The Increase/Decrease keyboard shortcuts can also specify a string of characters, so that the level is increased/decreased when any one of the keys is pressed.
- **Reset To Level:** Specifies a keyboard command that resets the Layer to a specified level. This does not automatically activate the Layer.



- **Randomize:** Specifies a keyboard command that sets the Layer to a randomly selected level.

- **Level Images:** Specify the image to be shown for each layer by double-clicking or selecting from the drop-down menu. An image can be left blank to display nothing for that level. Using transparency in the images can be very useful.
- **Level Name:** Each level can be given an individual name, which is used to change the name of the piece for reporting purposes during play. The level's name either replaces the piece's normal name, or else modifies the piece's normal name as a prefix or suffix.

Examples of Layers

- *For a basic two-sided counter, add a Layer, and select an image that represents the reverse side. Change **Activate** to **Flip** and set the key to Ctrl-F.*
- *To represent fatigue in an Army counter, give it a Layer named* Fatigue. *Select **Always Active**, choose four images that represent the levels, and change* Increase *to* Increase Fatigue *and* Decrease *to* Decrease Fatigue. *A* Reset *command named* Rest *using Ctrl-R could be used to bring the Army counter back to full strength. Name the levels " (fatigue 1)", and so on, and check **is suffix** to append the current fatigue level to the piece's name.*

Composite Piece Images

In most cases, a Game Piece image is a static representation, based on a single created or scanned image.

However, you can construct the appearance of a Game Piece using a composite set of images. For the basic piece image, you could use a solid-color (or even transparent) GIF or PNG, and then create the actual piece appearance by compositing semi-transparent Layers. This gives you more flexibility when creating actual units, as well as cutting down on the number of graphic images you require, as you can re-combine image layers to create the pieces.

Even if you use this method, the Basic Piece Trait for the piece must still be assigned an image. The image can be a transparent or semi-transparent PNG or GIF.

For example, we create the Russian armies for our World War II game. Each Russian Tank unit will consist of a red background, one Layer consisting of a Tank icon, and another Layer showing the unit strength. Because there are two kinds of Tank units, one light and one heavy, each will have a different strength, which is determined when the unit is deployed. We define the units as follows:

- Basic Piece Trait image includes the solid red background.
- One Layer, called Icon, shows the Tank icon. (Everything else in the image is transparent except the tank icon, so the red background will show through.)
- Another Layer, called Strength, has two levels, and each shows the unit strengths for light and heavy tanks. (As above, the rest of the image is transparent except the Strength text.)

Game Pieces: Trait Descriptions

When a tank is deployed, the player can select the layer showing the correct strength of the unit. The counter will appear to be a single image. Such a scheme could easily be implemented by using Prototypes (see page 67).

Layer Trait Properties

The Layer Trait includes these system Properties. <layer_name> is the **Name** of the Layer defined in the **Layer** dialog box.

Property	Description
<layer_name>_Image	Name of the currently active level's image file.
<layer_name>_Name	Name of the currently active level.
<layer_name>_Level	Number of the current level.
<layer_name>_Active	Has a value of true if the Layer is active, false otherwise.

EXAMPLE: A Layer named Manpower that is active and showing level 4 defined with image Man04.gif and name (strength 4) would have the following Properties:

- *Manpower_Image = Man04.gif*
- *Manpower_Name = (strength 4)*
- *Manpower_Level = 4*
- *Manpower_Active = true*

These Properties could be used in a Global Key Command to automatically remove all counters whose manpower was zero.

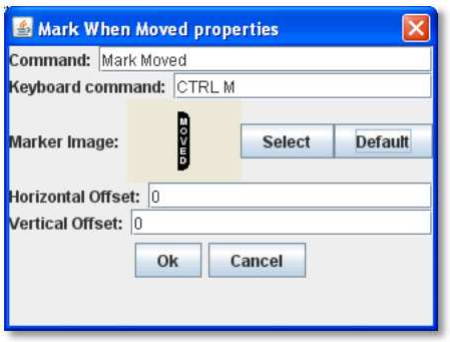
NOTE: The Game Piece Layer Trait is not related to the Game Piece Layers option for Map Windows.

Mark When Moved

A piece with the *Mark When Moved* Trait will display a specifiable image every time they are moved. Specify the image and the position at which to draw the image. You can also toggle the image on and off manually.

In order to enable this feature, you must also go to the Global Options of the module and enable the setting **Mark pieces that move**. Enabling this feature will automatically add a button to each Map Window, which when clicked will clears the Moved status of all pieces on the map.

The Mark When Moved Trait is a requirement for the Movement Trail Trait.



The Trait has these attributes:

- **Command:** Menu text of the command used to manually mark piece movement. (Even if left blank, the keyboard command will still appear on the Command Menu.)
- **Key Command:** Keyboard shortcut of the command to manually mark piece movement.
- **Marker Image:** Image displayed to mark piece movement. Click **Select** to choose a custom image.
- **Horizontal Offset:** Horizontal offset, in pixels, of the displayed image.
- **Vertical Offset:** Vertical offset, in pixels, of the displayed image.

Mark When Moved can be very useful in PBEM games, which may take days or longer between turns, to keep track of opponent moves.

Mark When Moved Properties

The Mark When Moved Trait includes one system Property:

Name	Description
Moved	Has a value of true if the piece has been moved.

Marker

A Marker sets (marks) one or more custom Properties on a Game Piece. The defined Property is static and its value cannot be changed during the game.

Setting a Property does not in itself give a Game Piece any particular behavior. The Property must be recognized by some other component in the module. Markers are used by Global Key Command and Game Piece Layers components and often by custom Java classes used in a module.

To use a comma in a name or value, precede it with a backslash ('\').

Defining Multiple Properties: You can define multiple name-value pairs for multiple Properties by separating the names and values with a comma (',').



The Trait has these attributes:

- **Property Name:** Name of the Property.
- **Property Value:** Value of the Property. Can be text or numeric.

For Properties that can be changed during a game, see *Dynamic Property* on page 46.

NOTE: The Marker Trait is not related to the Place Marker Trait.

Assigning a Piece to a Game Piece Layer

Marker Traits are commonly used to assign Game Pieces to Game Piece Layers (GPLs), which cause Game Pieces to be drawn on different levels. (You should set up the Game Piece Layers for the map first. See page 24 for more information.)

To assign a Game Piece to a Game Piece Layer,

1. Set up the Game Piece Layers for the map.
2. Assign the Marker Trait to the piece.
3. In **Property Name**, type the name of the Game Piece Layer Property (for example, Layer).
4. In **Property Value,** type the name of the layer you will assign the piece to. The name must match one of the GPLs already assigned to the map.

Totaling the Number of Pieces on a Map

You can use the Marker Trait in conjunction with the Set Global Properties Trait to sum the number of pieces on a map.

1. Create a Global Property called PieceTotal.
2. For the new Global Property, create a Change-Property button called Zero Total that will set PieceTotal to 0 (In **Type**, choose *Set Value Directly*).
3. Create a Marker on each piece you want to add to the count. Name the Marker *Count*, with a value of 1.
4. Create a Set Global Property Trait on each piece, which will increment PieceTotal by 1.
5. Create a Global Key Command called Total Pieces. For Global Key Command, use the keyboard shortcut of the Set Global Property Trait you specified in Step 3, and in Matching Properties, enter *Count = 1*.
6. Create a Toolbar Action Button called *Count Report*. In **Display Message**, and enter *Total Number of Pieces on Map:* $PieceTotal$.
7. Create a Multi-Action Button called Total, and add the Zero Total, Piece Total, and Count Report buttons to it.

Now, when the Multi-Action Button is clicked, PieceTotal will first be zeroed out (to remove any previous totals), then each piece will add 1 to the PieceTotal, and the Action Button will report the total in the chat window.

Mask

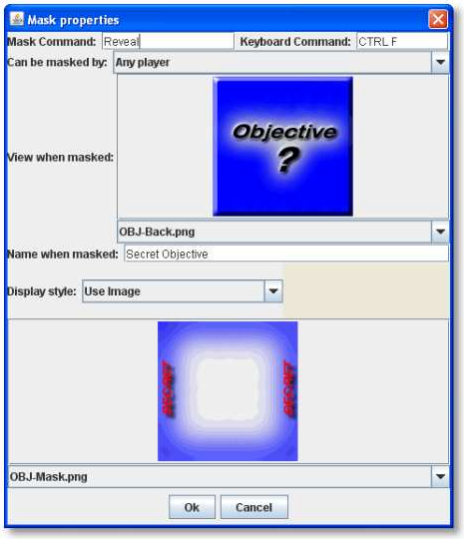
A Mask is used for hiding the true appearance of a piece, such as when you play a facedown playing card. A Masked Game Piece will show its mask to players other than the one who hid it. The hiding player can still view its true face. This Trait is useful for card games, block games, or games with concealable pieces. (Note that unlike an Invisible piece, a Masked piece will still remain visible.)

Any piece with a Mask Trait, such as a playing card, must have a back side image defined, or when the masked Game Piece is revealed the Piece will seem to vanish to all players.

Like the Invisible Trait, this Trait only hides Traits that appear before it. Generally, it should be before any Invisible Trait and after all other Traits of the piece.

Use of the Mask Trait will require you to define Sides in the game. (See page 37 for more information.)

A piece with the Mask Trait is "owned" by the player who masks it. If unmasked and masked again by a different player, the second player becomes the owner. Menu commands of Traits hidden by a masked piece are not available to non-owning players. A setting in the Global Options determines whether or not non-owning players can unmask pieces.



A Mask Trait is best used only once for a given piece. For pieces with that may have several different appearances, use the Layer Trait instead. See page 49.

The Trait has these attributes:

- **Mask Command:** The name of the Command Menu entry that mask or unmasks this piece.
- **Key Command:** The keyboard command to mask or unmask this piece.
- **Can be Masked By:** Defines who may mask the piece from other players)

o. *Any Player* means that any player may mask this piece, including observers.

o. *Any Side* means that any player who has been assigned a Side in a game (not an observer) can mask this piece. If the player resigns and another player takes the Side, then the new player for that Side will be the owner.

  o. *_Any of the Specified Sides* enables you to enter a list of Sides. Only players assigned to one of the named Sides can mask the piece, but the players of all the listed Sides will be able to see and modify the piece. This is useful for referee players or games with multi-player teams.

    - **View when Masked:** To non-owning players, the piece will be drawn using this image.
    - **Name when Masked:** To non-owning players, the piece will be given this name.
    - **Display Style**: Determines how the owning player sees a masked piece. The following options are available:

p. *Inset* draws the regular piece with the mask image at reduced size in the upper left corner. (The

size of the reduced image is not customizable.)

o. *Background* draws the mask image at full size and the regular piece at reduced size centered within it. (To make a mask image appear in a different location, use a mostly-transparent graphic the same size as the counter or Card, with the mask in the location that you want it to appear.)

o. *Plain* draws only the mask image, so the piece looks the same to all players. A **Peek** command key may be specified. When the owning player selects the **Peek** command, he will see the unmasked piece so long as it remains selected (that is, until he clicks elsewhere on the map). If the **Peek** command key is left blank, then the owning player will see all selected pieces in their unmasked state.

*A **Peek** command is temporary. If you□d like to allow the owning player to see the hidden piece on a permanent basis, use one of the other display styles instead.*

o. *Use Image* draws the unmasked piece and then a specifiable image on top of the piece. The image should make use of transparency to let some of the piece information through.

EXAMPLE: An ordinary playing Card can be implemented by setting the Basic Piece Trait to represent the front of the Card. Then add a Mask Trait. In the Mask Trait settings, specify an image for the back of the playing Card. When a player types Ctrl-P, that Card will be known only to him (as though held in his hand). Typing Ctrl-M will reveal the Card to the other players (as when playing it on the table).

Mask Properties

The Mask Trait includes one System Property:

Property	Description
ObscuredToOthers	Has a value of true if the piece is masked.

Move Fixed Distance

The Move Fixed Distance Trait defines a command to move the piece a fixed distance upwards and to the right.

If this piece has a Can Rotate Trait listed *before* this Trait, then the resulting direction will be relative to the current facing of the piece.

- If a Game Piece had the Can Rotate Trait followed by Move Fixed Distance (upwards 60 pixels), then the Move Fixed Distance command would move the piece in whatever direction the top of the piece is facing.
- If a Game Piece has Traits Move Fixed Distance (upwards 60 pixels), followed by the Can Rotate Trait, then the move command would move the piece towards the top of the screen regardless of the facing of the piece.

The Trait has these attributes:

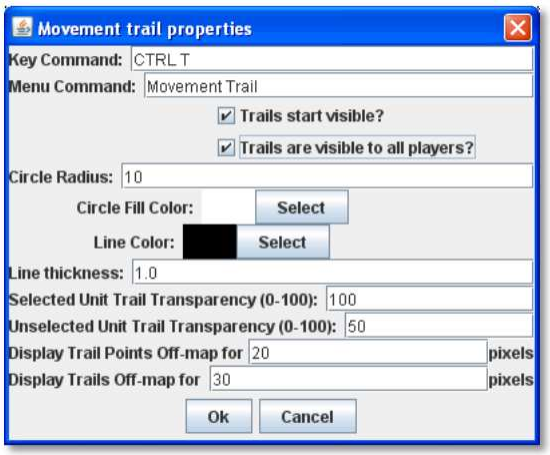
- **Description:** Description of the command (will not appear on the piece).

- **Command Name:** Menu text of the command used to move the fixed distance.
- **Keyboard Shortcut:** Keyboard shortcut of the command used to move the fixed distance.
- **Distance to the Right:** Distance, in pixels, the unit is moved to the right. To move the unit to the left, use a negative number.
- **Distance Upwards:** Distance, in pixels, the unit is moved up. To move the unit down, use a negative number.
- **Move Entire Stack:** If selected, when the piece is part of a stack that is not expanded, the command will move the entire stack.
- **Advanced Options:** If selected, additional movement increments can be specified. The two numbers specified in the advanced options are multiplied together, and added to the basic distance, to get the final distance moved. *Example: An army unit can conduct a forced march for extra movement. The amount of additional movement depends on its supply, which is tracked by a Dynamic Property. The Move Fixed Distance Trait is given an additional offset of one hex times the value of the supply level Property.*

Movement Trail

Game Pieces with the Movement Trail Trait will leave behind a graphical trail showing the positions through all positions to which the piece has been moved. The trail consists of a circle for each past location, connected by straight lines. The piece must also contain a Mark When Moved Trait.

The Movement Trail is reset when the moved status of the Mark When Moved Trait is cleared.



The Trait has these attributes:

- **Key Command:** The keyboard shortcut to toggle the movement trail. If left blank, then the trail is always visible.
- **Menu Command:** The Command Menu item to toggle the movement trail. If left blank, no menu entry appears, although the keyboard command may still be enabled.
- **Trails Start Visible:** If selected, at the beginning of each move, the trail will be visible.
- **Trails Visible To All Players:** If selected, then toggling the visibility of the trail will affect all players' views and will be saved along with the game. Otherwise, each player controls the visibility of trails on that player’s view.

- **Circle Radius:** The radius, in pixels, of the circle representing each location in the trail.
- **Circle Fill Color:** The color of the location circles.
- **Line Color:** The color of the connecting lines.
- **Line Thickness:** The thickness, in pixels, of the connecting lines.
- **Selected Transparency:** The transparency of the trail when the piece is selected. 0 is invisible; 100 is opaque.
- **Unselected Transparency:** The transparency of the trail when the piece is not selected. 0 is invisible; 100 is opaque.
- **Display Points Off-Map:** If the map has buffer space surrounding the boards, the trail circles will be drawn within this distance from the board edges.
- **Display Trails Off-Map:** If the map has buffer space surrounding the boards, the trail lines will be drawn within this distance from the board edges.

Movement Trails can be very useful in PBEM games, which may take days or longer between turns, to keep track of piece movement in detail.

Automatically Resetting Movement Trails

Using several commands together, you can cause movement trails to be automatically reset on a Game Piece at the start of each turn.

1. Add the Movement Trail and Mark When Moved Traits to the piece (or Prototype) for which you wish to automatically reset trails.
2. Add a Global Key Command to the module. Assign it a Hotkey. For **Matching Properties**, enter Moved = true. For **Global Key Command**, enter the Key Command from the Mark When Moved Trait (which will toggle the movement trail).
3. Create a Turn Counter and a Counter (or List). Add a Turn-Based Global Hotkey. Use the Hotkey of the Global Key Command you created in Step 2.

Now, each time you advance the Turn Counter, the Global Hotkey will trigger the GKC, which will reset movement trails on any pieces that have been moved. The trails will show again normally when the piece is moved.

Non-Rectangular

The Non-Rectangular Trait enables you to specify an arbitrary shape for a Game Piece, based on a partially transparent image such as a GIF or PNG file.

The shape of a Game Piece is used to determine where the player must click to select a Game Piece or bring up its Command Menu. It also is used to highlight the outline of the piece when it has been selected.

By using transparent colors in your GIF or PNG, you can make your Game piece be drawn with any shape. However, without the Non-Rectangular Trait, the piece can be selected even by clicking on the transparent portions of the image, which can lead to confusion if the image uses a great deal of

transparency.

The Trait has one attribute:

- **Image Shape:** select an image shape from the drop-down list of existing image files in your module, or double-click to add a new one.

Place Marker

A Game Piece with the Place Marker Trait will have a menu command that places a different piece (the *marker*) on or near it.

You can select any existing piece for the marker, or define a new one from scratch.



The Trait has these attributes:

- **Horizontal Offset:** The marker will be placed this many pixels to the right of the original piece. Any value other than zero will prevent the marker from stacking with the original piece.
- **Vertical Offset:** The marker will be placed this many pixels above the original piece. Any value other than zero will prevent the marker from stacking with the original piece.
- **Match Rotation:** If selected, and both the original piece and the marker have the Can Rotate Trait, then the rotation angle of the marker will be adjusted to match that of the original piece.
- **Place Marker:** Choose whether the marker should be place on the top of this piece's stack, on the bottom, or directly above/below the triggering piece.
- **Keystroke to apply after placement:** Optional keystroke to be applied automatically to the marker immediately after being placed

EXAMPLE: If a game uses a fortification counter to indicate fortified status of an army counter, this Trait could be given to the army counter to place a fortification marker on the army with a keyboard command, as an alternative to dragging the fortification counter from the Game Piece Palette.

The Place Marker Trait is not related to the Marker Trait.

Play Sound

The Play Sound Trait enables you to specify a command that plays an audible sound. The Trait has

these attributes:



- **Menu Command:** The name of the menu item in the Command Menu.
- **Key Command:** The Keystroke or Named Command for the command.
- **Sound Clip:** Select a file in .au, .aiff, or .wav format to add it to the module. The sound file specified in this field will be played when the action is invoked. (MP3s are currently not supported.)
- **Send Sound to Other Player:** If selected, then the sound will be echoed to other players when playing live or reading from a logfile. Otherwise, the sound is only audible to the player who invoked the command.

Playing a Sound with a Piece Action

To combine a Play Sound trait with another piece action, create a Trigger Action that includes the Play Sound Trait with the piece action.

For example, a Zorkon war cruiser has a Cloaking Device represented by an Invisible Trait. Each time the war cruiser cloaks (or de-cloaks), we want it to play a “whoosh” sound. We first define the Invisible Trait no command name, but with a shortcut of Ctrl -I. Next, we define a Play Sound trait with no command name, a “whoosh” sound clip, and a shortcut of Ctrl-P. Finally, we define a Trigger with the command name *Cloak* and a shortcut of Ctrl-Shift-C. Under **Perform These Actions**, we enter Ctrl-I and Ctrl-P. Now, selecting Cloak from the war cruiser□s Command Menu will invoke both Traits.

Alternately, for simple actions, instead of defining a Trigger Action, you can specify the keyboard command for the Play Sound trait to use the same keyboard command for the other action. When this keyboard command is invoked, both Traits will be triggered.

Movement Sounds

Using the Play Sound Trait, you can cause a Game Piece to make a sound each time it is moved in a particular Map Window, simulating the sound of a game piece being moved on a board.

1. Locate or create the sound file you wish to play when the piece is moved. (Typically, this is a “click” sound.)
2. Create a Game Piece with the Play Sound Trait. Specify a keyboard command. For **Sound Clip**, select the sound file you created in Step 1.
3. Double-click the **[Map Window]** node the sound will be played on.
4. In the Map Window Properties dialog, in **Key Command to Apply to All Units Ending Movement on This Map**, enter the keyboard command for the Play Sound Trait you defined in

Step 2. Now, each time the piece is moved, the sound clip is played.

Property Sheet

The Property Sheet Trait attaches an arbitrary set of editable Properties to a Game Piece. This can be used for character sheets, piece attributes, and many other functions. The Trait has these attributes:

- **Menu Command:** Name of the menu item to show the Property Sheet window.
- **Key Command:** Keystroke or Named Command to show the Property Sheet window.
- **Commit Changes On:** When a player edits the Properties window during play, there are three methods for committing changes:
  - *Commit on Every Keystroke:* Every keystroke and tick-mark click you make are immediately committed as you make them. Other players see your changes immediately.
  - *Commit on Apply Button or Enter Key:* Changes are not communicated to other players until you click the **Apply** button at the bottom of the Property Sheet, press the Enter key on your keyboard, or close the Property Sheet window.
  - *Commit on Window Close or Enter Key:* Changes are not communicated to other players until you press the Enter key or close the Property Sheet window.
- **Background Color:** You may customize the background color of each Property Sheet window, for example to use different colors for the pieces belonging to different Sides.
- **Properties:** You may select from these formats in which to display Properties:
  - *Text:* A simple, single-line field that accepts text.
  - *Multi-line text:* A field that accepts multi-line text. This type of field stretches to fill extra space on the Property Sheet window. It is suitable for free form notes.
  - *Label Only:* This is not really a Property; it simply adds text to your Property Sheet. It is useful for documenting your Property Sheet.
  - *Tick Marks:* Displays one or more rows of checkboxes. Suitable for tracking ammo or damage. Players specify a current and maximum value range.
  - *Tick Marks with Max Field:* As above, but the maximum value is displayed in an editable field to the left of the checkboxes. Suitable for role-playing games where damage tracking is based on a character attribute.
  - *Tick Marks with Value Field:* As Tick marks, but the current value is displayed in an editable field. Suitable for large-value Properties where clicking ticks might be impractical and when the exact tick value is important. For example weapons that track 100+ rounds of ammo.
  - *Tick Marks with Value and Max:* As Tick marks, but both current value and maximum values are editable.
  - *Spinner:* A numeric Property that includes increment and decrement buttons.

Using Tick Marks

Tick Mark Property types have a value and a maximum. Either, both, or neither may be displayed

as a text box in addition to the tick marks. Initially, the maximum and value are both 0, so no tick marks appear. To set the value or maximum when the box is not shown, right-click in the area where the tick marks would appear.

Pre-defining Values in a Property Sheet

Generally, Property Sheets values are defined at game time. For example, in a game where pieces represent fantasy gladiators, the Property Sheets will be used to record each individual fighter's personal attributes like Strength or Hit Points, and are filled in by the players when the game begins.

However, you can pre-define the values in a Game Piece's Property Sheet, so that the selected piece will have the values filled in already. This is useful when all pieces of a given type have the same Property Sheet values. For example, in the fantasy gladiator game, we decide that every Orc has a Strength of 12 and 14 Hit Points. If these values were pre-filled, each Orc counter's Property Sheet would have these values already assigned when placed on the map.

Note that this method will not work if the piece inherits a Property Sheet from a Prototype. The Game Piece must have the Property Sheet Trait directly in order to be pre-defined.

To pre-define a Game Piece's Property Sheet,

1. In the Game Pieces Palette, select the piece whose Property Sheet you want to pre-define. (Do not drag it to the map.)
2. In the Palette, right-click the piece and select the Property Sheet from the Command Menu.
3. Enter the values for the sheet as desired.
4. Save the module. Whenever a Game Piece of this type is drawn from the palette, the values you entered will be already defined in the Property Sheet.

Prototype

The Prototype Trait assigns a Prototype to the piece from the module's list of Prototype Definitions. A Game Piece can have any number of Prototypes assigned.

Before assigning a Prototype to a Game Piece, define it under the **[Prototype Definitions]** node. See page 67 for more information on creating Prototypes.

In terms of Trait order, a Prototype Trait is treated as a single block of Traits. Traits below the Prototype will affect all Traits that are part of the Prototype. Traits that are part of the Prototype will affect all Traits above the Prototype.



The Prototype Trait has one attribute:

- **Prototype Name:** The name of a Prototype Definition.

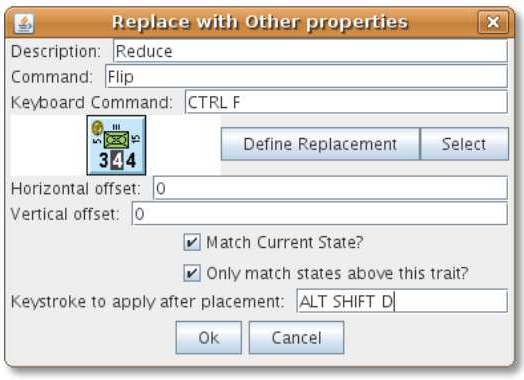
A Game Piece assigned a Prototype exposes a Property called Type.

Replace with Other

A Game Piece with the *Replace with Other* Trait will have a menu command that replaces the piece with a different one. You can select any existing piece for the replacement, or define a new one from scratch.

For example, a unit that can be destroyed but still leaves a wreck behind, could be given this Trait to replace the original counter with a wrecked version. This would be more convenient than dragging a new piece from the Game Piece Palette, and can't be accidentally undone, as a Layer Trait could.

- **Description:** Description of the Trait. (Will not appear on the piece.)
- **Command:** Text of the menu item used to replace the piece.
- **Key Command:** Keyboard command of the menu item used to replace the piece.
- **Define Replacement/Select:** Click **Define Replacement** to define a new replacement for the piece, or click **Select** to select an existing piece.
- **Horizontal Offset:** The replacement will be placed this many pixels to the right of the original piece.
- **Vertical Offset:** The replacement will be placed this many pixels above the original piece.
- **Match Current State:** If selected, VASSAL will attempt to put the replacement piece in the same state as the original piece. Layers will be set to the same level, labels will be given the same value, rotation angles will match, and so on. The state of a particular Trait will carry over only if it has an exact match in the replacement, that is, the Properties settings of that Trait are the same in both the original and replacement piece.
- **Only Match States Above this Trait:** If selected, VASSAL will only replace states in Traits that occur above this one in the list Traits in the Game Piece Editor. For example, the state of a Marker that's above this one will change if the state in the new Game Piece. If it's below, then it will not change if the new Game Piece has the same marker Property.
- **Place Marker:** Choose whether the marker should be place on the top of this piece's stack, on the bottom, or directly above/below the triggering piece.
- **Keystroke to Apply After Placement:** If desired, enter a keystroke to be applied to the replacement piece after it is placed. For example, the replacement Wreck counter described above includes a Play Sound Trait (Ctrl-P) of an explosion, which is applied after the Wreck counter is placed.



Replacing a Piece with Multiple Pieces

The *Replace With Other* Trait will replace a Game Piece with only a single piece. To replace a Game Piece with multiple pieces, combine the Replace with Other Trait with a Trigger Action. For example, we want to replace a Game Piece A with 3 copies of piece B. On Piece A, we define a Replace with Other Trait with a Keyboard Shortcut of Ctrl-R. (We leave the **Command** empty.) The Replace with Other Trait will replace A with B.

We define a Trigger Action on Piece A, with a shortcut of Ctrl-T. In the **Perform These Keystrokes** section of the Trigger Action, we enter Ctrl-R three times, once for each copy of B.

Now, invoking the Trigger Action on Piece A with Ctrl-T will perform Replace with Other three times, replacing A with three copies of B.

The same process could be used to replace multiple pieces with non-identical pieces. However, we would need to define three different Replace with Other Traits on Piece A, one for each piece type. The keyboard shortcuts for each of these Traits would then be included in the Trigger.

Creating Lockable Pieces

Using Replace With Other, you can create pieces that can be moved normally on the board, but will include a command that locks them, preventing them from being moved. (What will actually be occurring is that a command will switch the mobile piece with the immobile one, and back again. However, to players, this switch will be invisible.) To do this, you need to create two nearly identical pieces, one mobile and one immobile.

1. *Create the mobile piece:* In a Game Piece Palette, create the mobile Game Piece. Add whatever Traits you choose to add to define the piece, except Replace with Other. During the game this piece will be moved normally, by drag and drop, to its position.
2. *Create the immobile piece:* In the Editor, right-click the Piece you just created, pick **Copy**, and then **Paste** the copied piece into the Palette. During the game, this piece will be locked, so add the Does Not Stack Trait to this piece. In the **Does Not Stack** dialog, define how this piece will be selected, and whether or not the piece can be moved when selected or not moved at all.
3. Go back to the mobile piece you created in Step 1, and add the Replace with Other Trait. Define the Properties of the Trait in the **Replace with Other** dialog as follows:
    - **Description:** Enter *Lock Command.*
    - **Command:** Enter *Lock*.

- **Key Command:** Enter Ctrl+L (or other appropriate shortcut).
- Click **Select**. Browse to, and select, the piece you created in Step 2 (the immobile piece).
- Check **Match Current State**.
- Leave the other values on the dialog empty and click **Ok**.

4. So the piece can be unlocked, for the immobile piece you created in Step 2, add the Replace with Other Trait, and then define the Properties of the Trait in the **Replace with Other** dialog. Then enter the following:
   - **Description:** Enter *Unlock Command.*
   - **Command:** Enter *Unlock*.
   - **Key Command:** Enter Ctrl+U (or other appropriate shortcut).
   - Click **Select**. Browse to, and select, the piece you created in Step 1 (the mobile piece).
   - Check **Match Current State**.
   - Leave the other values on the dialog empty and click **Ok**.

During a game, players can select the mobile piece, move and place it normally, and then select the **Lock** command. This will replace the mobile piece with the immobile one. To unlock the piece later, players choose **Unlock** on the immobile piece, which invokes the replacement (mobile) piece.

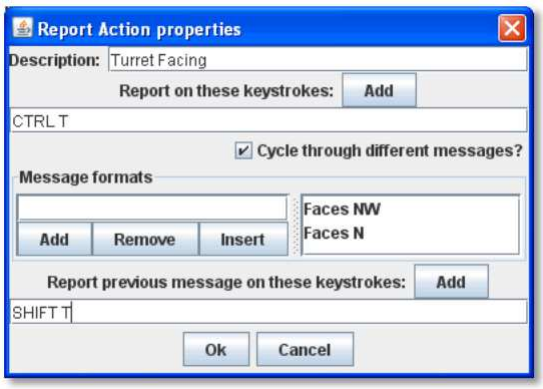
Alternately, instead of defining the immobile piece on a game piece palette in Step 2, you can define it in the Replace with Other Trait of the mobile piece by clicking **Define Replacement** instead of **Select** in Step 3. Then, add the exact same basic image and Traits as the mobile piece possesses, as well as the Does Not Stack Trait. Add and define the Replace with Other Trait as well, using the parameters from Step 4. This method will make the immobile piece inaccessible through game palettes, and only accessible by selecting the **Lock** command on the mobile piece.

Report Action

A Game Piece with the Report Action Trait will report a configurable message to the Chat Window when any of a given set of key commands is entered. (The report will appear whether the key commands are entered by a player or invoked automatically, such as with a Global Key Command.)

In order for Report Actions to display text in the Chat Window, the Map Window that the piece currently is on must have the setting **Auto-Report Format For Units Modified on This Map** enabled. (By default, this is set to $message$.) If this setting is empty, then no Reports will be returned.

- **Report on these Keystrokes:** Specifies the keys that this Trait will respond to. Click the **Add** button to specify more than one key.
- **Cycle through Different Messages:** If left unchecked, the same message will be reported whenever any of the above keys are pressed. If selected, the message to be reported will cycle through the list specified below. Each time one of the keys if pressed, the next message in the list will be reported, returning to the beginning after the end is reached.
- **Report Format:** The Message Format for reporting non-cycling messages:



o. menuCommand is the name of the piece's Command Menu item that corresponds to the control key pressed.

o. oldPieceName is the name of the piece before the action is applied.

o. newPieceName is the name of the piece after the action is applied.

o. mapName is the name of the map where the piece is located.

o. oldMapName is the name of the map before the action.

o. location is the map location where the piece is located.

o. oldLocation is the location before the action is applied.

If a Game Piece is deleted or replaced as the result of an action, then the value of oldLocation and oldMapName will depend on the order of the Traits, while mapName and location will be blank.

- **Message Formats:** A list of Message Formats for cycling messages. Available variables are the same as above. Any Properties defined on the piece will be substituted. To access the value of a Property before the change, add the prefix *old* to the name. For example, if a Game Piece has a Property hitPoints, then $hitPoints$ gives the value after the key command and $oldhitPoints$ gives the value before.
- **Report Previous on these Keystrokes:** When any of these keys are pressed, the message reported will be the one the precedes the last reported message, instead of the following one.

Report Action Examples

- An Infantry unit has a single layer that is activated with a Ctrl-F "Flip" command. You add a Report Action with Report Key Ctrl-F, and a message *$newPieceName$ flips in hex $location$*. When the player flips the unit, the Chat Window reports *Infantry flips in hex 3321*.
- A piece includes the Invisible Trait, toggled by Ctrl-I. A Report Action Trait is added with report key Ctrl-I and two cycling messages: *$oldPieceName$ goes invisible in $location$* and *$newPieceName$ revealed in $location$*. The messages will be shown in order whenever will report when the unit becomes invisible or is revealed.

Restrict Commands

The Restrict Commands Trait enables you to disable or completely remove certain keyboard commands from a Game Piece when certain conditions or contexts apply. For example, you could restrict some commands on a piece to only be useable by certain players, or on certain boards. Like other Traits, it will only affect those above it in the Properties list, so it should be placed after the commands it restricts.



Some uses of the Restrict Commands Trait include:

- A piece may remove certain commands based on where it is on the map by matching the CurrentZone Property.
- A piece with a Layer specifying a damage level may disable commands based on the Level Property of that Layer.
- Commands used only during the setup portion of the game (Turn 0) can be disabled during gameplay turns (Turn > 1).

The Trait has these attributes:

- **Name:** A name, for identification purposes.
- **Restriction:** Select *Hide* to remove a command from the Command Menu entirely. Select *Disable* to disable (gray out) the command. In either case, the restricted action will not be invoked with its corresponding keyboard combination is pressed.
- **Restrict when Properties Match:** The commands will be restricted when the Properties of this piece match the given expression
- **Restrict These Key Commands:** Specify the keyboard commands that will be hidden or disabled. The corresponding Command Menu item (if any) will also be restricted.

*A Restricted Command will not be fired as part of a Trigger Action or GKC if the Properties of the piece match the restricting conditions. For example, if a piece's Clone command (Ctrl-C) is Restricted on a board named Battlefield, (*CurrentBoard = *Battlefield), then any Trigger using that Ctrl-C command will not work correctly for pieces on the Battlefield board.*

Restricted Access

A Game Piece with Restricted Access can only be controlled by a specified Side. Other players will not see menu items corresponding to Traits appearing above the Restricted Access Trait in the list of Traits for the Game Piece, and the corresponding keyboard commands will do nothing.



The Restricted Access Trait has these attributes:

- **Belongs to Side:** Enter a Side, and then click **Add** to add it to the list of Sides. The Sides must be one of those listed in the definition of

Player Sides. Only players playing one of the specified Sides will be able to modify this Game Piece.

- **Also Belongs to Initially-Placing Player:** If selected, then the player who initially clicks on the piece (or first places it on any map) will become the owner, in addition to listed Sides. It is a good idea to specify at least one Side when using this option. Otherwise, any pieces created by an observer will not be able to be removed. If, during a game, a player clicks the **Retire** button to become an observer, then all pieces owned by that player become owned by nobody, even if the player was already an observer. Pieces in a Game Piece Palette can be manipulated by anybody, as long as no game is in progress.

If you assign the Restricted Access Trait to a Game Piece, you will need to add Sides to the game. See page 37 for more information.

Return to Deck

The Return to Deck Trait will send a Card to a Deck. This Trait will have no effect on ordinary Game Pieces, only Cards. The Trait has these attributes:

- **Menu Command:** Menu text of the command used to send the piece to a Deck.
- **Key Command:** Keystroke or Named Command used to send the piece to a Deck.
- **Select Deck**: Click **Select Deck** to choose a Deck to be sent to. Alternatively, select **Choose Destination Deck at Game Time**, and players will be prompted to select a Deck to send the Card to after invoking the command.

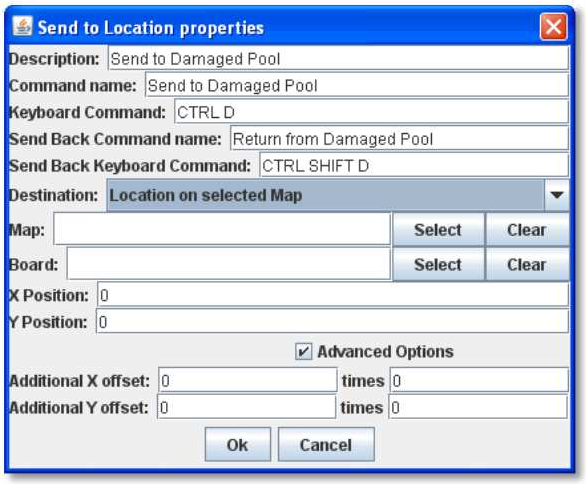
```
For example, in a game in which Cards are drawn from a Deck, used, and placed into
a discard pile, both the Deck and the discard pile will be represented by a Deck
component. By adding a Return to Deck Trait to each Card, with the text *Discard*
and the command 'Ctrl-D', and the Discard Pile selected as the destination, then
clicking Ctrl-D on any Card would automatically send it to the Discard Pile.
```

This Trait□s name is slightly misleading. A Card with this Trait can actually be sent to any *Deck: the one the Card came from, or an entirely different Deck.*

For more about creating Decks and Cards, see page 74.

Send to Location

The Send to Location Trait adds a command that moves a Game Piece directly to another location. The Trait has these attributes:



- **Command Name:** Text of the menu item used to send the piece.
- **Key Command:** Keyboard shortcut of the menu item used to send the piece.
- **Send Back Command Name:** Menu text for an undo command, which will return the piece to its original location.
- **Send Back Keyboard Command:** Keyboard shortcut for the undo command
- **Destination:** Choose a destination type for the piece.

o. *Location on Selected Map*: Sends the piece to a defined X-Y coordinate. In **Map**, click **Select**, and then select a Map Window. In **Board**, click Select, then select a Board from the selected Map Window.

Further, specify the X-Y coordinates of the location on the board, in pixels. If no board is specified, positions are relative to the Map Window.

o. *Zone on Selected Map*: (Used for maps with Zones defined.) In **Map**, click **Select**, and then select a Map Window. In **Zone Name**, enter the name of a Zone from the Map Window.

o. *Region on Selected Map*: (Used for maps with Regions defined.) In **Map**, click **Select**, and then select a Map Window. In **Region Name**, enter the name of a Region from the Map Window.

o. *Another Counter, Selected by Properties*: To send the piece to another counter, in **Property Match**, specify one or more Properties to match as a final destination for the piece. The Property Match should match a unique piece or unexpected results may occur. For example, to send the piece to a unique piece on the Main board named *Commander*, the value of **Property Match** would be CurrentBoard = Main && PieceName = Commander.

*Game Pieces that are moved to another Map by the Send to Location Trait will not trigger the **Auto-Report Format For Movement To This Map** message on the new Map.*

- **Advanced Options:** The value of these two Message Formats will be multiplied together and

added to the position specified above to give the final destination position for the piece.

EXAMPLE: A game may require that damaged units be returned to a Damaged pool for repairs. Different boxes in the pool represent the amount of time before the unit is fully repaired. A Game Piece may be given a Send to Location Trait with name Send to Damaged Pool and command Ctrl-P and position corresponding to the first box of the pool, with an additional offset, determined by the level of a Layer representing the damage, to place it in the appropriate box in the pool.

Set Global Property

The Set Global Property Trait enables a Game Piece to change the value of a Global Property. The Trait has these attributes:

- **Description:** A descriptive name of the command (Will not appear in the Command Menu).
- **Global Property Name:** The name of the Property to be set. (The name can include the name of another Property. Set it off by using $-signs; for example $Example$_Property would be a valid Global Property name.)
- **Locate Property Starting in the**: You may name a Zone or Map containing the Global Property to be set, or you may set the Property based on the piece's current location, looking for the occupied Zone or Map before defaulting to the Module. (The name can include the name of another Property. Set it off by using $-signs; for example $Example$_Map would be a valid Map or Zone name.).
- **Is Numeric:** If selected, then the value of the Property will be restricted to integer values.
- **Minimum Value:** Numeric values will be restricted to no less than this number.
- **Maximum Value:** Numeric values will be restricted to no more than this number.
- **Wrap Around:** If selected, then when incrementing this numeric Property, values will wrap around from the maximum to the minimum (or vice versa).
- **Key Commands:** Adds any number of commands to the right-click drop-down menu for this Game Piece. Click the **New** button to add a new command. For each command, specify the text of the drop-down menu entry and the keyboard shortcut. The type defines how the Property value should change:

o. *Set value directly:* Sets the Property to a fixed value, after substituting values of other Properties defined for this Game Piece.

o. *Increment numeric value:* Adds a fixed value to the Property. You can use a numeric value or the value of another Property. (If you specify a Property, enter the name of the Property in $-signs; for example, $ExampleProperty$.)

o. *Prompt user*: Displays a dialog for the user to type in a new value.

o. *Prompt user to select from list:* Displays a dialog with a drop-down menu for the user to select from.

Spreadsheet



The Spreadsheet Trait attaches an editable table of data to a Game Piece. A Spreadsheet is simply for the tabular display of data (or text). It is not capable of performing any mathematical operations on cells, rows, or columns in the table.

- **Menu Command:** Text of the menu item used to display the Spreadsheet.
- **Key Command:** Keyboard shortcut of the menu item used to display the Spreadsheet.
- **Number of Rows:** Number of rows in the spreadsheet.
- **Number of Columns:** Number of columns in the spreadsheet.

Currently, VASSAL has no method for handling mathematical formulas.

Pre-populating Spreadsheet Data

Generally, Spreadsheet values are defined at game time. However, you can pre- define the values in a Game Piece□s Spreadsheet, so that the selected piece will have the values filled in already. This is useful when all pieces of a given type have the same Spreadsheet values.

Note that this method will not work if the piece inherits a Spreadsheet from a Prototype. The Game Piece must have the Trait directly in order to be pre-defined.

To pre-define the values of a Game Piece's Spreadsheet,

1. In the Game Pieces Palette, select the piece whose Spreadsheet you want to pre-define. (Do not drag it to the map.)
2. In the Palette, right-click the piece and select the Spreadsheet command.
3. Enter the values for the sheet as desired.
4. Save the module. Whenever a Game Piece of this type is drawn from the palette, the values you entered will be already defined in the Spreadsheet.

Sub-Menu

The Sub-menu Trait enables you to group menu items associated with other Traits into a sub-menu in the Game Piece's Command Menu. Use it to organize command menus for ease of use.



Sub-menus may contain other sub-menus, to any nesting level. Items added to a Sub-menu will not appear independently.

Items added to a Sub-Menu are case-sensitive.

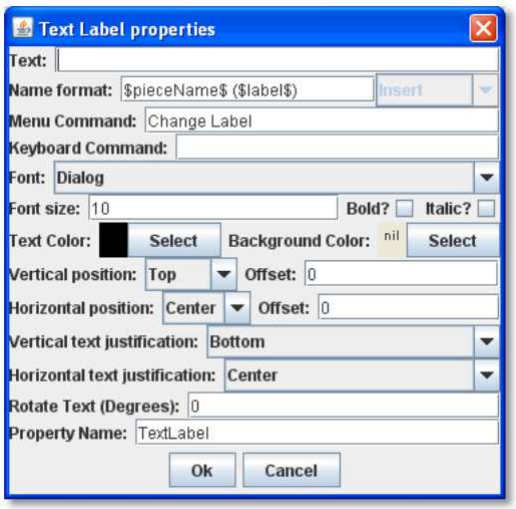
The Trait has these attributes:

- **Menu Name:** Name of the sub-menu.
- **Sub-commands:** Click **Add** to add the name of another command from the piece's Command Menu. Commands added will be displayed in the Sub-menu in the order they are listed.

Example: If a Game Piece has three separate layer Traits with corresponding activate commands Entrench, Fortify, and Blockade, then those menu items can be gathered under a single sub-menu named Defense by creating a Sub-Menu Trait with Menu Name Defense *and Sub-commands* Entrench, Fortify, *and* Blockade.

Text Label

The Text Label Trait displays a text label on or near the Game Piece. The text of the label can be fixed or specifiable by a player at game time. The Trait has these attributes:



- **Text:** The starting value for the label text. (You can set this to the value of a Property on the piece by enclosing it in $-signs.) By enclosing the text within tags, you can use simple HTML format to specify various colors, fonts and sizes. Example: <html>Bold text<p>with a line break<p>and different colors</html> would display as:

Bold text

with a line break

and different colors

- **Name Format:** A Message Format that specifies how the name of this piece will be reported: pieceName is the name of the piece excluding the label, label is the value of the label text (including, unfortunately, HTML tags). If the label is empty, then the default name of the piece is always used.
- **Menu Command:** If not blank, gives the text of the corresponding menu item in the piece's Command Menu
- **Menu Key Command:** If blank, the text of the label is permanent. If set, then gives the keyboard command to set the text of the label.
- **Font:** Text is drawn using this font.
- **Font Size/Bold/Italic:** The text is drawn at this size, optionally in bold or italics.
- **Text Color:** The text is drawn using this color.
- **Background Color:** The text is drawn within a solid rectangle of this color. Click **Select** and then **Cancel** to use a transparent background.
- **Vertical Position:** Draw the label with the given offset from the top, bottom, or center of the piece.
- **Horizontal Position:** Draw the label with the given offset from the left, right, or center of the piece.
- **Vertical Justification:** Whether the top edge, bottom edge, or center of the label will be drawn at the Horizontal Position specified above.
- **Horizontal Justification:** Whether the right edge, left edge, or center of the label will be drawn at the Vertical Position specified above.
- **Rotate Text:** The text will be rotated clockwise by this angle. Rotation is performed after the horizontal/vertical justification and positioning specified above.
- **Property Name:** The value of this label will be exposed as a Property with the given name. Ordinarily, a Text Label trait comprises its own Property, which you can name when you create the Trait.

  ```
  For example, in a naval wargame, we want a Text Label trait to show each ship's
  individual name, such as _HMS Victory_. We use the following settings:
  ```

- **Text:** We leave this blank. Players will be able to specify the string at the start of a game.
- **Name Format:** $pieceName$ ($label$). For a battleship piece, this would show, *Battleship (HMS Victory).*
- **Menu Command:** *Set Ship Name*, with a keyboard shortcut of Ctrl+N.
- (We set font size, color, position, and other cosmetic settings as appropriate for the piece.)
- **Property Name:** ShipName. We can now use ShipName as a Property for things like Report Traits on the piece, and other functions.

Using a Text Label to Display a Property

You can use a Text Label Trait on a Game Piece to display the value of any Property defined on the Game Piece (as well as Global Properties). This is handy to display Property values you have separately specified for the Game Piece, such as Dynamic Properties, or to create a "display piece" that shows the value of some Global Property.

To display the value of a separately defined Property as a Text Label,

1. In both **Text** and **Name Format**, enter the name of the piece's Property or Global Property you wish to display, surrounded by $-signs. (You can add additional label text here; any text not surrounded by $-signs will be displayed literally.)
2. Leave the values of **Menu Command**, **Keyboard Command**, and **Property Name** blank.
3. Enter display values (font size, color, position, and justification) for the label as appropriate.

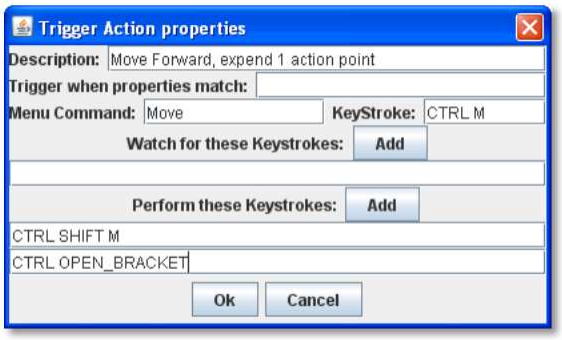
For example, if a Starship piece had a Dynamic Property Trait called EnergyLevel, we could display the value of the starship's

Energy Level with a Text Label by entering this in both **Text** and **Name Format**:

Ship's Power: $EnergyLevel$.

When the ship has a power level of 5, what will be shown in the label is *Ship's Power: 5*

You can enter any number of Property names. For example, you could also show the ShieldLevel Property in the label by entering this in **Text** and **Name Format**:



Ship's Power: $EnergyLevel$ Shield Level: $ShieldLevel$.

Trigger Action

A Trigger Action Trait combines multiple keyboard commands into one, or automatically invokes keyboard commands in response to other keyboard commands, when certain conditions apply. A Trigger Action can be keyed to fire on command, to fire when one or more keystrokes are made, or when either of these apply and certain conditions (Properties) are matched.

The Trait has these attributes:

- **Trigger Name:** Descriptive name of the Trigger Action. (Will not appear on the Command Menu.)

- **Trigger When Properties Match:** The corresponding key commands will be performed only if the piece matches this Property expression. Property match is optional.

  ```
  A keyboard command must be actively invoked to launch a Trigger Action. Trigger
  Actions cannot passively listen for matching Properties and then fire
  automatically.
  ```

- **Menu Command:** Adds an item to the piece's Command Menu that will launch the trigger commands manually commands, as long as the Property expression is matched.
- **Key Command:** Keystroke or Named Command for the manual menu command.
- **Watch for These Key Commands:** After the user types any of these Keystrokes or Named Commands, the commands listed under **Perform These Keystrokes** will be launched, if the Property expression is matched.
- **Perform These Key Commands:** The key commands to be invoked after one of the above key commands is observed and the Property expression is matched. The commands are invoked in sequence from top to bottom.

Example 1: A piece has a Layer to track action points and a Move Fixed Distance Trait to move it forward. The Move Fixed Distance Trait can be assigned the key command Ctrl- SHIFT-M with no command name (so that it does not appear in the Command Menu). Then a Trigger Action Trait with the command Move and the keystroke Ctrl-M can trigger both the Move command and decrease the action points layer by one.

Example 2: A piece has separate Layer Traits for hit points and for a "critically wounded" status for when the hit points are less than 2. A Trigger Action Trait can watch for the keystrokes that affect the hit-point layer and respond by activating the wounded layer by matching the Property expression for when the hit points are < 2 and the wound level is not active.

To suppress the command menu labels for the keyboard commands that compose the Trigger Action, omit the text labels for the individual commands. Then the only way to invoke these commands will be to fire the Trigger Action. If the text labels are not omitted, then each individual command will also appear in the piece's command menu separately.

Prototype Definitions

Most games have counters that look different from one another, but behave very similarly. *Prototypes* are a method of using templates to vastly simplify the definition of most Game Pieces. A Game Piece assigned a Prototype will inherit all of the Traits assigned to the Prototype.

You first define a Prototype as a set of Traits, and then use the set in the definition of other Game Pieces. A Game Piece can use any number of Prototypes.

For example, in the module *World War II*, we define a Prototype called Tank. The Tank Prototype includes these Traits: Mark When Moved, a Layer showing the tank with its regular strength and strength when damaged, and a Text Label allowing the owner to assign an ID number to the tank. Then, in the US pieces list, we create a counter for the Sherman tank, and in the German pieces list, we create one for the Tiger tank. Both of these pieces are assigned the Trait *Prototype – Tank*. Both Sherman and Tiger tank counters will include all the Traits from the Tank Prototype, without having to assign the Traits individually to each Tank counter.

It's strongly recommended that you use Prototypes whenever possible when creating Game Pieces, as module updates, revisions, and maintenance become considerably easier. When you decide to change or update the pieces in a module, updating the relevant Prototype will update all the pieces with that Prototype as a Trait, without having to individually update each piece.

For example, players of the World War II module described above complain that the font used in the Text Label Trait on the tanks is too small. It□s easy to edit the Tank Prototype to change the Text Label Trait to enlarge the font size from 9 points to 12. All pieces with the Tank Prototype will follow suit automatically, without having to update the pieces individually.

Pieces that use Prototype Traits can still have additional Traits defined to give them their own unique behavior.

Defining a Prototype

The process for defining a Prototype is just the same as for defining a Game Piece, but omits the Basic Piece Trait. Trait order applies to Prototypes in exactly the same way as for Game Pieces.

To create a new Prototype Definition,

1. In the Configuration window, right-click the **[Game Piece Prototype Definitions]** node and pick **Add Definition.**
2. In the **Prototype** dialog, select the Traits you wish to assign to the Prototype Definition, like you would for a Game Piece.
3. Click **Ok**.

You can now assign the Prototype to Game Pieces or Cards by assigning them the Prototype trait, and specifying the new Prototype.

Using Prototypes

- As with regular pieces, Traits in Prototypes are evaluated from bottom to top.
- Prototypes can include other Prototypes. Just assign the included Prototype as a Trait in the containing Prototype.
- You can use Prototypes to store any group of Traits that need to be re-used across multiple piece types as a block. For example, imagine a war game where there are both land and naval pieces. The naval pieces represent a variety of ships and submarines, all of which have common Traits: a Text Label Trait indicating the unit□s speed, and a Delete Trait named Sink that removes the piece from the game. We could create a Prototype called Naval Unit that includes each of these Traits and assign the Prototype to all ship and submarine pieces.
- Pieces in At-Start Stacks, or that were created as part of the Replace with Other or Place Marker Traits, which use Prototypes will be updated if the Prototype definition is ever changed.
- Game Pieces that use Prototypes and are part of a saved game will not be updated if the Prototype definition is updated. As a result, a Prototype Definition can change in a later version of a module without invalidating saved games from previous versions. However, if you wish the pieces in the saved game to use the updated Prototype, you will need to run the Saved Game Updater in order to bring the pieces in a Saved Game up to date. See page 91 for more information.

Pre-Setting Traits in a Prototype

If you want to pre-set pieces in a Game Piece Palette to start in a certain state, you can use key commands on them to change their state, and then save the module. (See page 40 for more information.)

Prototype Definitions: Defining a Prototype

However, if the Game Pieces use Prototypes, the Game Piece Palette always loads with the default state of the Prototype. To resolve this, you can pre-set the state of a Trait in the Prototype Definition.

To pre-set one or more Traits in a Prototype Definition,

1. In the **[Prototype Definitions]** node, right-click the Prototype Definition you wish to pre-set Traits for.
2. In the **Properties** dialog, in the image preview window, right-click the piece image. From the Command Menu, select a Trait and adjust it as desired.

   ```
   If the Prototype does not include an image preview, you will need to expand the
   size of the *Properties* dialog in order to select the Command Menu. Right-click
   where the piece would be shown, in the white background area.
   ```

3. Repeat Step 2 for each additional Trait you wish to pre-set.
4. Click **Ok**.

Game Piece Image Definitions

Using Game Piece Image Definitions, you can build your own Game Piece images, by combining text, images, and standard NATO military symbols. Images defined in this component will be available for use with Game Pieces, just like any externally created images you have imported into the module.

You can use your own images instead of the computer-drawn NATO symbols, so for many games, you will be able to define the whole counter set with just a handful of images. Furthermore, you can change the size and layout of all the counters in your game easily by adjusting the layouts.

Set up a Game Piece Image Definition in two steps:



I. **Layout:** First, create a Game Piece Layout. In the Game Piece Layout, you specify the position, size and style of all items to be drawn on the counter. Colors, actual text, and symbol selections are made in step 2.

AI. **Define Pieces:** Define an individual image using a layout. In each Image Definition, you specify the actual colors, text and symbols to be used for that image, based on the layout.

After defining the Game Piece Image, you can assign it to the	**Some examples of Game Piece Image Definitions**	
Game Piece, along with any Traits.		

Use of Game Piece Image Definitions to create counters is optional. It□s best for use in wargames or other games with a large number of standardized counters.

Game Piece Image Elements

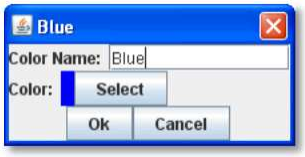
Game Piece Image Definitions have the following elements: Named Color, Font Style, and Layout.

Named Colors

Each color you wish to use in Image Definitions is predefined and given a name. These colors will appear in a palette for selecting foreground or background colors in the image. 14 Standard colors are built-in: Clear, White, Black, Light Gray, Dark Gray, Red, Green, Blue, Orange, Pink, Cyan, Magenta, Yellow.

- **Color Name:** The name of the color that will appear in drop-down menus in the Image Definitions.
- **Color:** Standard Color selector to select the color to be associated with the name.

To create a named color,



1. Right-click the **[Named Colors]** node and pick **Add Named Color.**
2. On the **Color Name** dialog, in **Color Name**, name the new color.
3. In **Color**, click the color selector, and select the desired color.
4. Click **Ok**.

Font Styles

Font Styles used in Image Layouts are defined here and selected by name from drop-down menus. A Font Style consists of a Font Family, size and style (plain, bold, italic, bold-italic). A default style is always defined: 12 point Dialog font.



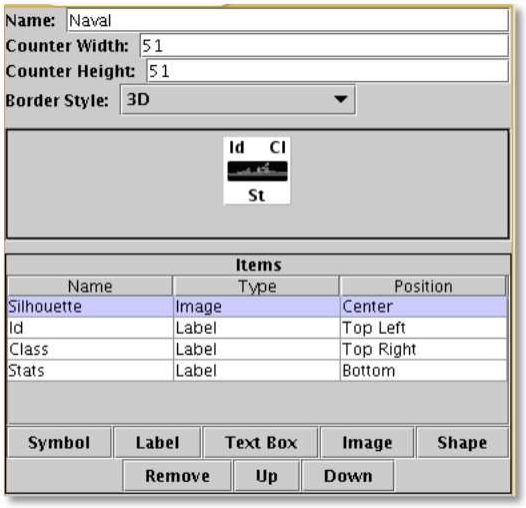
- **Style Name:** The name of the Font Style that will appear in drop-down menus in the Image Layout.
- **Font Family:** The Font Family to use. To ensure maximum compatibility and portability, only the pre-defined Java logical fonts are available as options.
- **Size:** The size of the font style in points.
- **Bold:** Click on to select a Bold font style.
- **Italic:** Click on to select an Italic font style.
- **Sample:** Display a sample of your selected font style.

To create a font style,

1. Right-click the **[Font Styles]** node and pick **Add Font Style.**
2. On the Font Style dialog, enter the settings for the font style.
3. Click **Ok**.

Game Piece Layouts

A Game Piece Layout defines the general appearance and positioning of the items used in drawing an image. Each Layout is composed of a rectangle of the background color, a border, and items such a symbols, labels, text boxes, images, or shapes. These items are generic and defined in terms of their appearance and position of the counter. Later, when the image is defined based on the layout, we can specify actual settings for each of these items, such as the specific symbol to use, or the specific text to show.



A Game Piece Layout has these attributes:

- **Name:** The name of the Image Layout.
- **Counter Width:** The width, in pixels, of all counters created using this layout.
- **Counter Height:** The height, in pixels, of all counters created using this layout.
- **Border Style:** The border style for all counters created using this layout. Border styles available are:

o. *Plain:* Single-pixel line of defined color.

o. *Fancy:* Two-pixel shaded line of defined color. Mild 3D effect.

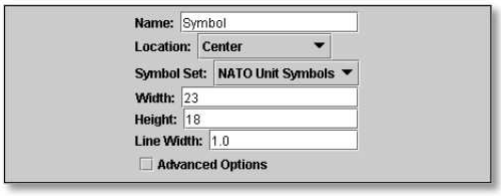
o. *3D*: A three-dimensional shaded border. Two pixels

wide, color automatically determined from background color.

o. *None:* No Border

  - **Symbol:** A Symbol is a generic symbol to be drawn by VASSAL. These must be NATO Unit Symbols. The particular symbol is chosen in the Game Piece Image.

p. **Name:** The name of the Item. Items must be uniquely named within an Image Layout.



o. **Location:** Select the location of the item on the counter.

o. **Symbol Set:** Select the Symbol Set to use. (The only symbol set available currently is standard NATO Unit Symbols.)

o. **Width:** The width of the body of the symbol in pixels.

o. **Height:** The height of the body of the symbol (not including the Size specifier) in pixels.

o. **Line Width:** The width of the line (in pixels) used to draw the symbol. Fractional line widths can be used. The lines are drawn with anti-aliasing turned on, to produce smooth looking lines of any width. When using a small symbol size, a line width of 1.0 will usually give the best results.

  o. ___ **Advanced Options:** If selected, you can specify values for X and Y offset, Rotation, and whether or not to anti-alias the image.

    - **Label:** A Label is a text label drawn in a particular font at a particular location. The value of the text can be specified in the individual images or in the layout, in which case all images using this layout share the same value.

p. **Name:** The name of the Item. Items must be uniquely named within an Image Layout.

o. **Location:** Select the location of the item on the counter. The location also determines the text justification, i.e. selecting Top Left ensures that the upper left corner of the text is in the upper left corner of the image. Once the justification is set by the Location, you can still use the X/Y offset in the advanced options to place the text in a different location.

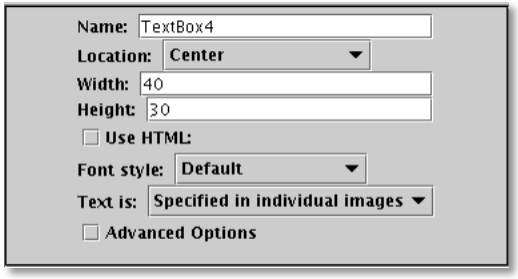
o. **Font Style:** Select the name of the Font Style to be used for this Text Item.

o. **Text is:** Select whether the text is specified in the layout or in the images.

  o. ___ **Advanced Options:** If selected, you can specify values for X and Y offset, Rotation, and whether or not to anti-alias the image.

    - **Text Box:** A Text Box Item is multi-line area of text drawn in a particular font at a particular location. The value of the text can be specified in the individual images or in the layout, in which case all images using this layout share the same value.

p. **Name:** The name of the Item. Items must be uniquely named within an Image Layout.



- **Location:** Select the location of the item on the counter. The location also determines the text justification, i.e. selecting Top Left ensures that the upper left corner of the text is in the upper left corner of the image. Once the justification is set by the Location, you can still use the X/Y offset in the advanced options to place the text in a different location.
- **Use HTML:** If selected, then the contents will be interpreted as HTML.
- **Font Style:** Select the name of the Font Style to be used for this Text Item.
- **Text is:** Select whether the text is specified in the layout or in the images.
  - ___ **Advanced Options:** If selected, you can specify values for X and Y offset, Rotation, and whether or not to anti-alias the image.
    - **Image:** An Image item is an imported image.

p. **Name:** The name of the Item. Items must be uniquely named within an Image Layout.

- **Location:** Select the location of the item on the counter.
- **Image is:** Specify whether the image is specified in this layout or in the images that use this layout. Use the File Open Dialog box to locate a copy of the image you wish to use on your PC. When you save the module, VASSAL will attempt to copy this image into the images folder within the module zip file. You can also manually copy images into your images folder.



- **Advanced Options:** If selected, you can specify values for X and Y offset.
  - **Shape:** A Shape Item is a simple geometric shape.

p. **Name:** The name of the Item. Items must be uniquely named within an Image Layout.

- **Location:** Select the location of the item on the counter.
- **Width:** Select the width of the shape.
- **Height:** Select the height of the shape.
- **Shape:** Select the type of shape.
- **Bevel:** For Rounded Rectangle shapes, larger bevel values mean rounder corners.
  - ___ **Advanced Options:** If selected, you can specify values for X and Y offset, and whether or not to anti-alias the image.
    - **Items List:** Items are drawn in the layout based on their order in this list. An item at the top of the list will be drawn on top of the items below it. An item below another item in the list will also be drawn below it in the Layout. Use these buttons to control items in the Items List:

p. **Remove:** Removes the selected Item.

Game Piece Image Definitions: Game Piece Images

o. **Up/Down:** These move the selected Item up or down in the list and cause the item to be drawn on top of, or below, the other elements.

Creating a Game Piece Layout

As you design the Layout, a preview is shown in the **Game Piece Layout** dialog box.

To create a Game Piece Layout,

1. Right-click the **[Game Piece Layouts]** node and pick **Add Game Piece Layout.**
2. On the **Game Piece Layout** dialog, specify the elements of the layout.

o. **Name**

o. **Counter Width and Counter Height**

  o. __ **Border Style**

3. Select one or more items to include in the Layout by clicking the corresponding button, and then entering the details of the item. Repeat for any additional items.
4. Click **Ok**.

Game Piece Images

Now that you□ve created a Named Color, Font Style, and a Game Piece Layout, you can create one or more images that use these elements, and choose specific values for the layout items.

For example, we can create a Game Piece Layout called British Unit. The Layout uses a brown background and a Symbol Item placed in the center of the Layout. When we define an image based on the Layout, we can choose a specific NATO symbol to appear in the Layout (for example, Cavalry or Infantry). We can then save each image we create and use them when we assign images to Game Pieces.

A Game Piece Image has these attributes:

- **Name:** Specify a name for the image definition. This is the name under which this image will appear in the image-selector drop-down menu in a Game Piece Trait's Properties.
- **Background Color:** Select a background color for the image from the drop down list of available colors.
- **Items:** The Items panel shows the configurable items that make up your image layout. Click on an item to display the configurable options for that item in the bottom display panel. There is a different display panel for each type of item.

Symbol Item Configuration

- **Unit Size:** Select the NATO Unit Size specifier from the drop-down menu.
- **1st Symbol:** Select the Primary NATO Symbol from the drop-down menu.

- **2nd Symbol:** Select the Secondary NATO Symbol from the drop-down menu.
- **Symbol Color:** Select the color used to draw the symbol lines.
- **Background Color:** Select the color to use for the background of the symbol body.
- **Size Color:** Select the color used to draw the Size Specifier drawn above the symbol body.

Label Item Configuration

- **Value:** Enter the text to display on the image.
- **Foreground Color:** Select the color to use to draw the text.
- **Background Color:** Select the color to use to draw a box behind the text.

Text Box Item Configuration

- **Value:** Enter the text to display on the image.
- **Text Color**: Select the color to use to draw the text.
- **Background Color:** Select the color to use to draw a box behind the text.

Game Piece Image Definitions: Game Piece Images

Image Item Configuration

Import an image to draw at the position specified in the layout.

Shape Item Configuration

- **Foreground Color:** Select the fill color for the shape.
- **Background Color:** Select the color for the shape's outline.

Creating a Game Piece Image

To create a Game Piece Image,

1. In the **[Game Piece Layouts]** node, select a Game Piece Layout with which to create an image.
2. Right-click the node and choose **Add Game Piece Image.**
3. In the **Game Piece Image** dialog, specify the **Name** and **Background Color.**
4. In the **Items** list, select an item and specify the details of the item for this particular image. Repeat for all additional Items in the list.
5. Click **Ok**.

Once you've created an image, it will appear in the image-chooser drop-down list alongside imported images. You can then assign the Game Piece Image to Game Pieces. See page 40 for information on creating Game Pieces.

Decks and Cards

Many games include decks of cards as part of play; in fact, some games consist *entirely* of decks of cards. In VASSAL, Cards are created as a special form of Game Piece. However, the Deck functionality can be applied to a range of Game Piece types besides traditional Cards, such as for any pieces whose number is fixed or which need to be drawn randomly.

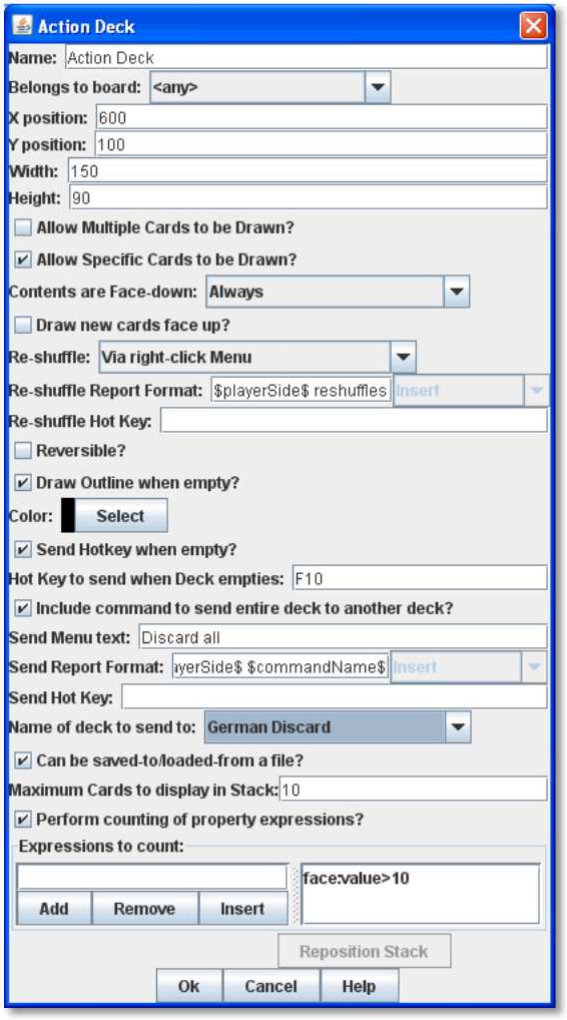
Creating a Deck

A Deck functions like a deck of playing cards. Each game begins with the contents of the Deck as specified in the **[Deck]** node. During a game, players may remove Cards from the Deck by clicking on the Deck and dragging cards from it with the mouse. This removes the Card from the Deck and assigns ownership to the dragging player. Dragging a Card onto the Deck area adds it back to the Deck.

Each Deck must be placed on a Map Window. A Deck may have a Command Menu that can include specialized commands that will affect only the Cards in the Deck.

Deck Attributes

Each Deck has the following attributes:



- **Name:** The name of a Deck is not used during game play. It is just used for identification in the module editor.
- **Belongs to Board:** If a name is selected, the Deck will appear on that particular Board. If a game does not use that Board, then the Deck will not appear. If *Any* is selected, then the Deck will always appear at the given position, regardless of the boards in use.
- **X, Y position:** The position in the Map Window of the center of the Deck. If this Deck belongs to a Board, the position is relative to the Board's position in the Map Window.
- **Width, Height:** The size of the "tray" holding the Cards. If the Deck is empty, this determines the area into which players may drag Cards to add them back to the Deck. It should be set to the same size as the Cards the Deck will hold.
- **Allow Multiple Cards to be Drawn:** Adds a Command Menu entry that prompts the user to specify the number of Cards to be drawn from the Deck with the next drag.
- **Allow Specific Cards to be Drawn:** Adds a Command Menu entry that prompts the user to examine the Deck and select exactly which Cards will be drawn from the Deck with the next drag.
- **When Selecting, List Cards Using:** When the user is prompted to select specific Cards from the Deck, individual Cards will be listed using the specified Message Format.
- **When Selecting, Sort Cards By:** When the user is prompted to select specific Cards from the Deck, the Cards can optionally be sorted (alphabetically) using the listed Property. Leave blank to list Cards by their occurring position in the Deck. Example: Cards in a Deck can use a Marker Trait to specify a Card number (001,002) and always list Cards in order of their assigned number.
- **Contents are Face-down:** Determines whether Cards in the Deck are always face-down, always face-up, or can be switched from face-up to face-down with a Command Menu entry.

o. **Face Down Report Format:** A Message Format that is echoed to the chat text window whenever a player selects the **Face Down** menu item (if enabled above): DeckName is the name of this Deck, commandName is the name of the menu item.

p. **Draw New Cards Face Up:** If selected, then Cards drawn from this Deck will be placed face-up on the playing area. If un-checked, then Cards in a facedown Deck are drawn face down and owned by the drawing player.

   - **Re-Shuffle:** If set to *Never*, then Cards remain in their original order; Cards are drawn from and added to the top. If set to *Always*, then Cards are always drawn randomly from the Deck. If set to *Via right-click menu,* then a **Shuffle** command is added to the Deck's Command Menu.

q. **Re-Shuffle Report Format:** A Message Format that is echoed to the chat text window whenever a player

selects the "Shuffle" menu item (if enabled above): DeckName is the name of this Deck, commandName is the name of the menu item.

- **Reversible:** Adds an entry to the Command Menu that reverses the order of Cards in the Deck.

  q. **Reverse Report Format:** A Message Format that is echoed to the chat text window whenever a player

selects the Reversible menu item (if enabled above): DeckName is the name of this Deck, commandName is the name of the menu item.

- **Draw Outline When Empty?** Whether to draw the "tray" for the Cards. The "tray" is a rectangle of the specified width and height, centered at the x, y coordinates. Only drawn when there are no Cards in the Deck, to indicate where to drag Cards to place them back in the Deck. May not be necessary if the Map Window contains a board onto which the tray is already drawn.
- **Color:** The color of the rectangle representing the "tray" above.
- **Send Hotkey when Empty?** Select this option to send a Global Hotkey whenever the Deck is emptied.

  r. **Hotkey To Send When Deck Empties**: Select the Hotkey combination to send whenever enough Cards are removed from the Deck to empty it.
- **Include Command To Send Entire Deck To Another Deck:** If selected, the Command Menu for this Deck will include a command that sends every piece in this Deck to a designated Deck. For example, this can be used to reshuffle a discard pile into its original Deck. The following three attributes all refer to this option.

  s. **Menu Command:** The text that appears in the Command Menu.

o. **Report Format:** A Message Format that is echoed to the chat text window whenever a player selects **Send**

to Another Deck (if enabled above): DeckName is the name of this Deck, commandName is the name of the menu item.

o. **Name Of Deck To Send To:** The name of the Deck that the contents will be sent to.

- **Can Be Saved-To/Loaded-From A File:** If selected, the Deck's Command Menu will include **Save** and **Load** items.

p. **Save** saves the contents of a Deck to a file.

q. **Load** replaces the contents of the Deck with the Cards specified in the file. Saved Decks can be loaded into an entirely different game than the one used to save the Deck. This option is useful for collectible Card games, in which a player may prepare a Deck offline in preparation for a game.

- **Maximum Cards To Be Displayed In Stack:** This defines the maximum number of Cards to graphically display in the Deck. The default is 10. For example, if set to 10, a Deck of 52 will appear to have 10 Cards, until the actual number of contents drops below 10. Then the Deck will visually start to shrink as Cards are removed. If set to 1, the Deck will appear flat like a single Card.
- **Perform Counting Of Property Expressions:** Enable processing of Property expression counting. Expressions must be defined.

r. **Expressions To Count:** Specify expressions to be counted within the Deck. These can be whatever you like and must be in the format of: <expression name> : <expression> For each expression, a map-level Property called <DeckName>_<expression name> is exposed. The exposed value is number of pieces for which that expression evaluates to true. An example of how to do this is provided on page 77. NOTE: Currently the only dynamic Property that can be

used in counting expressions is playerSide. Other dynamic Properties will most likely not update if they change after pieces move into a Deck.

- **Reposition Stack:** Click to drag a representation of the Deck to its final position on the board. This overrides any values you specified for X and Y positions, above.

Repositioning an Empty Deck:* You must have at least 1 Card defined for a Deck in order to use the **Reposition function. If the Deck does not have any Cards, like a discard pile, define a single dummy Card for the Deck, reposition the stack by dragging, and then delete the dummy Card when you're done.*

Decks and Cards: Creating Cards

First create the Deck, and then create the individual Cards in it.

To create a Deck,

1. Select (or create) a Map Window where your Deck will reside.
2. Right-click the selected **[Map Window]** node and pick **Add Deck.**
3. In the **Deck** dialog, enter the attributes for your Deck.
4. Click **Ok**.

You may now create the Cards for the new Deck.

Deck Properties

Decks include these Properties. <Deckname> is the name of the Deck.

Name	Description	Property Level
<Deckname>_numPieces	Number of Cards in the Deck.	Map
<Deckname>_<type>	Number of Card types in the Deck.	Map

Creating Cards

You create Cards like other Game Pieces, and may use any of the standard Game Piece Traits. However, by default, Cards include a Mask Trait to reflect their back face, which is hidden from view until revealed.

The term “Card” is used to describe any piece in a Deck, even if it does not necessarily resemble a traditional playing Card.

Cards may represent actual cards, blocks, map tiles, or any number of other counter types.

Once created, a **[Card]** node may not be converted into a **[Game Piece]** node, and vice versa.

In VASSAL 3.1.16 and earlier, Cards were created as part of a Deck and could never be pasted into Game Piece Palettes. Similarly, ordinary Game Pieces could not be pasted into Decks. This is no longer

true in versions 3.1.17 and later—the two types of piece are interchangeable between Palettes and Decks.

To create Cards for your Deck,

1. Expand the **[Map Window]** node where the Deck resides.
2. Right-click the new **[Deck]** node and pick **Add Card.**
3. In the **Card** dialog, select the Traits for the Card as you would a Game Piece.

Remember to define a base image for each Card, or the Card may appear to vanish when drawn from a Deck.

4. Click **Ok**.
5. Repeat steps 2-4 until all Cards have been added to the Deck.

In the Module Editor, Cards are treated as a distinct piece type. Cards may not be pasted into Game Piece Palettes, and ordinary Game Pieces may not be pasted into Decks.

For more information on creating Game Pieces, see page 40.

Cards and Prototypes

Cards from the same Deck often behave identically and are different only in their front faces. For instance, they most likely have the same Card back images, and will likely be sent to the same Deck (such as a discard pile) after use.

As a result, it□s highly recommended to define a Prototype for each Card type in your game, and then assign the relevant Prototype Trait to each Card in a Deck. (See *Prototypes* on page 67.)

For example, the game includes an Event Deck where the Cards describe random game events. You can create a Prototype called Event Card that includes a Mask Trait to reflect the common back of the Event Cards, and a Return to Deck Trait that sends discarded Event Cards back to the Event Deck.

By default, new Cards include a Mask Trait. You can delete the default Mask Trait and define it in the Prototype instead.

Copying and Pasting Cards

Copying and pasting Cards can vastly speed up the process of Card creation. Define the first Card, then right -click, **Copy** the Card, and paste it into your **[Deck]** node. You will now have an identical copy of the first Card. You can then edit the copy and select a new image for the face of the Card. You can create many new Cards quickly by repeating this method.

Editing the Contents of a Deck

You can make wholesale changes quickly to the entire contents of a Deck. Right-click the Deck and pick **Edit All Contained Pieces**. The **Properties** dialog for the first Card is displayed, but any

changes you make in the Properties dialog will affect all Cards in the Deck. Add, remove or edit Traits as usual, then click **Ok**. Your changes are applied to all Cards.

Card Properties

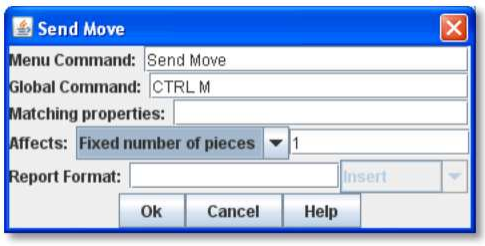
Cards have all the same Properties as regular Game Pieces. However, they also include these system Properties:

Name	Description
ObscuredToOthers	Has a value of true if the Card is masked.
DeckName	Name of the Deck the Card is currently stacked in, if any.

Deck Global Key Command (GKC)

This component adds an action that applies a key command to pieces contained within the Deck, similar to the Global Key Command component of a Map Window. Each Deck GKC has these attributes.

- **Menu Command:** Name of the Command Menu item.
- **Keyboard Command:** Keyboard shortcut of the menu item that initiates the command.
- **Global Command:** The key command that will be applied to the Cards in the Deck.
- **Matching Properties:** The key command will only be applied to pieces with the specified Properties. If you do not enter a Property expression, then all Cards in the Deck will be selected.
- **Affects:** The Global command can apply to all Cards in the Deck, or to a set number only. Use a setting of 1 to select the top Card.
- **Report Format:** A Message Format that is echoed to the chat text window whenever the Global Key Command is activated.



To add a Deck Global Key Command to a Deck,

1. Expand the **[Map Window]** node where the Deck resides.
2. Right-click the new **[Deck]** node and pick **Add Deck Global Key Command.**
3. In the **Deck Global Key Command** dialog, specify the behavior of the GKC.
4. Click **Ok.**

Card Decks in Practice

The following examples of possible Card Decks illustrate a variety of uses for them.

- **Playing Cards:** An ordinary Deck of playing Cards for Poker or Hearts would be set to: Allow Multiple = false, Allow Specific = false, Face Down = Always, Re-shuffle = Always, Reversible = false.
- **Discard Pile:** A Discard Pile is a type of Deck that is typically empty at game start. Cards from another Deck are drawn, played and then sent to the Discard Pile. When the other Deck is empty, the Discard pile is usually re-shuffled into the other Deck, and play continues. To create a typical Discard pile, define a Deck as usual, but use these settings:

o. Allow Multiple = false, Allow Specific = false, Face Down = Never, Re-shuffle = Never, Reversible = false.

o. Select **Include Command To Send Entire Deck To Another Deck** and define a command that when selected, will move all the discards back to the main Deck.

o. The Discard pile should begin empty, so there is no need to define Cards for it.

p. To move discards to the discard pile, for each Card in the main Deck, add a Return to Deck Trait, specifying the Discard Pile as the destination Deck.

If discards are not intended to return to the main Deck but instead are permanently removed from the game, it may be better to use the Delete Trait for each card instead of creating a Discard Pile

- **Force Pool:** A strategic game in which a nationality has a fixed force pool of variable-strength Infantry, Armor, and other forces can be modeled by making a Map Window representing the force pool, with a Deck of Infantry counters, a Deck of Armor counters, and so on. The Decks would be set to Allow Multiple = false, Allow Specific = false, Face Down = Never, Re-shuffle = Never, Reversible = false.
- **Random Turn Order Cards:** If the game has a random turn order, players may draw from a Deck to determine who moves first, second and third. Create a Deck where each Card is labeled 1, 2, 3, and so on. Select Allow Multiple = false, Allow Specific = false, Face Down = Always, Re-shuffle = Always, Reversible = false.
- **Playing Cards with Number of Cards Displayed:** You want to create a Deck of playing Cards, and display the number of red Cards, the number of black Cards, the number of face Cards, and the total number of Cards in the Deck. Create the Deck, and check **Perform counting of expressions**. Add the expressions of "red: Color = red" and "black: Color = black". Also add the expression "faceCards: value > 10". When creating your Cards, give them a Marker Trait named Color with the values of *red* or *black*. Also give your Cards a Marker Trait named Value with the numeric value of the Card. Then, you can refer to the counts with the map-level Properties of <Deckname>_red, <Deckname>_black, and <Deckname>_faceCards. The total can be referenced by the map-level Property of <Deckname>_numPieces.

Map Tiles

Some games make use of map *tiles*, which are usually shuffled at the beginning of a game, drawn

randomly and then placed to provide a random map layout. (If the layout is not random, or is in a regular row-column pattern, then an ordinary Map Window, with multiple Boards, will probably meet your needs better.)

To create randomly-placed Map Tiles, do the following:

1. Create (or select) a Map Window in which the tiles will be placed.
2. Create a solid-color board in the Map Window that will be large enough to accommodate your map tile layout.
3. Add one or more Game Piece Layers to the Map Window. The lowest layer should be the Tile level.
4. Create a Deck for the random Map Tiles to be drawn from. The Deck should have these settings: Allow Multiple = false, Allow Specific = false, Face Down = Always, Re-shuffle = Always, Reversible = false.
5. Create each Map Tile as a Card in the Deck. Assign each Tile to the Tile Game Piece Layer you created in Step 3.

Now, at game start, players can draw random map tiles and place them in the Map Window. The map tiles will always appear beneath all the other Game Pieces.

Dealing Random Cards to a Board

The Deck shuffle function can be used to deal randomly drawn cards from a Deck to pre-defined locations on the board, using a single click of a Global Key Command button.

To deal random cards to a board, do the following:

1. Create or select a Map Window to send the pieces to.
2. In the Map Window, create a board with an Irregular Grid. Label the Grid points numerically (1, 2, 3, 4, and so on).
3. Add a command to the selected Map Window (in **Key Command to Apply to All Units Ending Movement on This Map**) of Ctrl-I.
4. Create a Global Property named GridLocation. This Global Property will be used to track the next point to send the piece to. It should have minimum value that is the same as the lowest-numbered Irregular Grid point (that is, 1), a maximum value that is the same as the highest-numbered Grid point, and **Wrap Around** selected.
5. Create a Deck (on the same or different board) and make sure **Re-shuffle** is set to *Always*.
6. Add the first Card to the Deck. Add a Send to Location Trait to this Card: Send to Board (the Board created in Step 2) and the Region (enter $GridLocation$ in the **Region** box).
7. Add a Set Global Property Trait to the Card, with a command that will increment the GridLocation Property by 1. Give the Set Global Property command a shortcut of Ctrl-I (for Increment). Note that this is the same hotkey we specified in Step 3.
8. Copy and Paste the first card repeatedly until you have the desired number of cards in the Deck. Edit each card as needed with graphics or Traits.

9. Create a Global Key Command (GKC) for the same Map Window where the Deck is (or for the **[Module]** node).
    - For **Matching Properties**, enter *DeckName* = <the name of the Deck you created in Step 5>.
    - For **Key Command** use the Hotkey for the Send to Location Trait you created in Step 6.
    - For **Within a Deck, Apply To**, select *Fixed Number of Pieces*, and then enter the number of Grid points you created in Step 2.

When clicked, the GKC from Step 9 will affect the designated number of Cards in the Deck, triggering each one's Send to Location command. The first random Card is sent to Grid location 1, which then increments GridLocation by 1. So the next Card is sent to Grid location 2. The process continues until all the cards are dealt.

This process will deal one Card to each location before stopping. Instead, if you want to deal out all the Cards in the Deck, with multiple Cards on each Grid point, in Step 9, for **Within a Deck, Apply To**, select *All Pieces* instead.

Generating Random Results

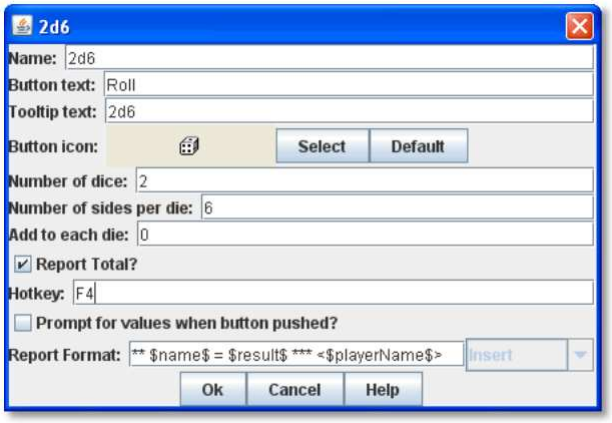
VASSAL has a variety of methods for generating random results: Dice Buttons, Symbolic Dice, and Random Text Buttons.

Dice Button

A Dice Button generates random numbers, simulating the roll of any number of dice of any number of sides. You may add any number of Dice Buttons to a module.

A Dice Buttons has these attributes:

- **Name:** The name of the dice button.
- **Button Text:** Text for the button in the Toolbar.
- **Tooltip Text:** Tooltip text for the Toolbar button.
- **Button Icon:** Icon image appearing on the button.
- **Number of Dice:** Number of dice rolled.
- **Number of Sides per Die:** Number of sides on each die.
- **Add to Each Die:** Number added to the roll of each individual die.
- **Add to Overall Total:** Number added to the total of all the dice.
- **Report Total?** If selected, results will total the dice. If not, the results of each die are reported individually (“2, 6, 3”).
- **Hotkey:** Keyboard shortcut for rolling the dice.
- **Prompt for Values when Button Clicked:** If selected, the player is prompted to enter the number, Sides and adds for the dice rolled after clicking the button. (If selected, you will not need to specify Number, Sides, and Adds as above.)
- **Report Format:** The Report Format specifies the Message Format for reporting the results: name is the name of the button as specified above, result is the result of the roll, nDice is the number of dice, nSides is the number of sides, plus is the modifier to each die, and addToTotal is the value added to the total.
- **Sort Dice Results:** Sorts the dice results.



To create a dice button,

1. Right-click the **[Module]** node and pick **Add Dice Button.**
2. In the dialog, specify the settings for the button.
3. Click **Ok**.

Dice Properties

Dice buttons include these system Properties. <name> is the name of the Dice button.

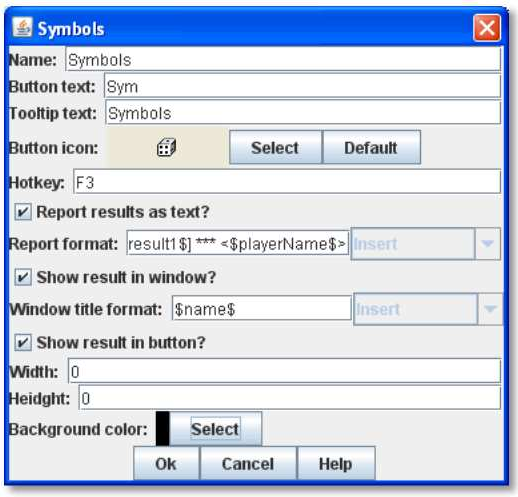
Property Name	Property Level	Description
<name>_result	Global	Value is the result of the Die Roll. Example: a Dice Button named 2d6 would include a Global Property named 2d6_result.

Symbolic Dice Button

A Symbolic Dice Button is used to define dice that use arbitrary images. When the button is clicked, a random face is selected for each Symbolic Die that this component contains. The results of the roll can be reported as text into the chat area, graphically in a separate window, or in the button itself.

Each button can roll any number of dice (represented by Symbolic Die components), each of which may have any number of faces (represented by Symbolic Die Face components).

- **Name:** The name of the Symbolic Dice Button.
- **Button Text:** Text for the button in the Toolbar.
- **Hotkey:** Keyboard shortcut for rolling the dice.
- **Report Results As Text:** If selected, report results to the chat area.
- **Report format:** A Message Format specifying the format for reporting text results: name is the name of the button as specified above, result1, result2, etc is the result of the 1st, 2nd, etc. Symbolic Die as configured below (replace the '#' symbol with the desired number), *numericalTotal* is the sum of the numerical values of the Symbolic Die rolls.
- **Show Result In Window:** If selected, show the results graphically in a standalone window.
- **Window Title Format:** A Message Format specifying the format for reporting results to the title bar of the standalone window.
- **Show Result In Button:** If selected, show the results graphically in the Toolbar button.
- **Width:** The width of the area for displaying results graphically.
- **Height:** The height of the area for displaying results graphically.
- **Background Color:** The background color to be used when displaying results graphically.



To create a Symbolic Dice Button,

1. Right-click the **[Module]** node and pick **Add Symbolic Dice Button.**
2. In the **Symbols** dialog, specify the settings for the button.
3. Click **Ok.**

After you define the symbolic dice button, you must define the actual dice rolled, including the dice faces.

Symbolic Dice

Each Symbolic Die has these attributes.

- **Name:** The name of the die.
- **Results Format:** A Message Format specifying how to report the result of this die roll. The resulting text will be substituted for result1, result2, and so on in the Symbolic Dice Button's results format: name is the name of this die as specified above, result is the text value of the Symbolic Die Face that is rolled, numericalValue is the numerical value of the Symbolic Die rolled.



To define a Symbolic die,

1. Right-click the **[Symbolic Dice Button]** node and pick **Add Symbolic Die.**
2. In the **Symbolic Die** dialog, specify the attributes of the die. Finally, you must define the face of each Symbolic Die.

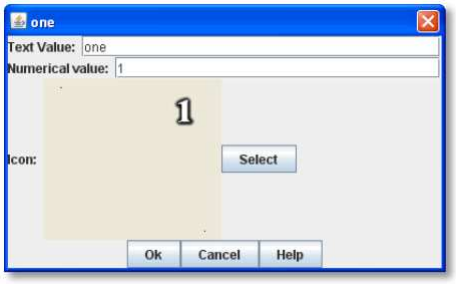
Symbolic Dice Faces

You must define the faces for each Symbolic Die. Each die face contains these attributes:

- **Text Value:** Text value is reported in the chat window.
- **Numerical Value:** You can assign a numerical value to the die face, if desired, which can be totaled when rolled.
- **Icon:** The die image shown in the separate window, or in the actual Symbolic Dice button.

To define a symbolic die face,

1. Right-click the **[Symbolic Die]** node and pick **Add Symbolic Die Face**.
2. In the **Symbolic Die Face** dialog, specify the attributes of the die.



*To quickly create multiple identical symbolic dice, first create one die, and define all its faces. Then, copy and paste the **[Symbolic Die]** node as many times as needed into your **[Symbolic Dice Button]** node.*

Symbolic Dice Properties

Symbolic Dice buttons include these system Properties. <name> is the name of the Symbolic Dice button.

Name	Property Level	Description
<name>_result	Global	Value is the result of the Symbolic Die roll. Example: a Symbolic Dice button named Ghost would include a Property named Ghost_result.

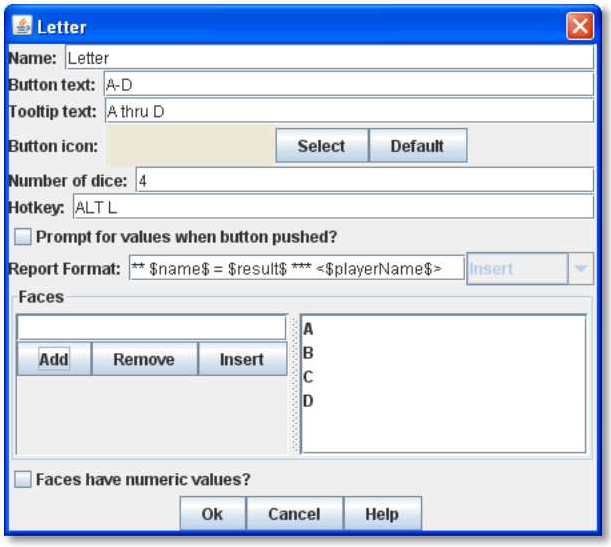
Random Text Button

A Random Text Button can be used to randomly select a text message from a list defined beforehand. For example, a button can be defined to select a random letter from the list A, B, C, or D.

It can also be used to define dice with irregular numerical values, such as a six-sided die with values 2,3,3,4,4,5, or dice with verbal values, such as a die with the results “Hit” or “Miss”.

One use for a Random Text Button could be to roll for results on a chart and then report the results to the Chat Window. However, such a chart roll may not have any modifiers applied.

A Random Text button has these attributes:



- **Name:** The name of the text button.
- **Button Text:** Text for the button in the Toolbar.
- **Tooltip Text:** Tooltip text for the Toolbar button.
- **Button Icon:** Icon image appearing on the button.
- **Number of Dice:** Number of dice rolled.
- **Hotkey:** Keyboard shortcut for rolling the dice.
- **Prompt for Values when Button Clicked:** If selected, the player is prompted to enter the number, sides, and adds for the dice rolled after clicking the button. (If selected, you will not need to specify Number, Sides, and Adds as above.)
- **Report Format:** The Report Format specifies the Message Format for reporting the results, which may include the following report variables:

$name$

Name of the button.

$result$

The result of the roll, either a list of the rolls, or the total of all rolls if the *Report Total* option is checked.

$nDice$

The number of dice rolled.

$nSides$

The number of sides on each die.

$plus$

The modifier added to each die.

$addToTotal$

The modifier added to the total.

$summary$

A summary of the roll results in the format `<value> x<count>[, <value> x<count>⋯]` (null where a roll generates no results).

- **Sort Dice Results:** Sorts the dice results.
- **Faces:** Specify the possible faces (results) for each die.
- **Faces Have Numeric Values:** If selected, enables the **Adds** and **Report Total** options.

o. **Add to Each Die:** Number added to the roll of each individual die.

o. **Add to Overall Total:** Number added to the total of all the dice.

o. **Report Total?** If selected, results will total the dice. If not, the results of each die are reported individually (“2, 6, 3”).

To create a random text button,

1. Right-click the **[Module]** node and pick **Add Random Text Button.**
2. In the dialog, specify the settings for the button.
3. Under Faces, enter the value for the first face, and click **Add**. The value is added to the list of results.
4. Repeat Step 3 until all faces have been added.
5. Click **Ok**.

Random Text Button Properties

Random Text buttons include these system Properties. <name> is the name of the Random Text button.

Name	Property Level	Description
<name>_result	Global	Value is the latest result of the Random Text button. Example: a Random Text button named Events would include a Property named Events_result.
<name>_summary_	Global	A summary of the most recent set of results (see Report Format, $summary$ variable)

Additional Module Components

This section discusses additional module controls. These controls are not available for individual Map Windows. Buttons associated with these controls will always appear in the Main Controls Toolbar.

Action Button

The Action Button combines a number of different actions into a single button. When the button is clicked, or its Hotkey is pressed, it can display a message to the Chat window, play a sound, or send a list of Hotkeys to other components.

An Action Button includes these attributes:



- **Description:** An identifying name for this button.
- **Button Text:** The text of the button to be added to the Toolbar.
- **Tooltip Text:** The tooltip text of the button to be added to the Toolbar.
- **Button Icon:** Icon for the Toolbar button.
- **Hotkey:** Keyboard shortcut for the Toolbar button.
- **Display Message?** Select to display a message to the Chat Window when the button is activated.
- **Report Format:** Message Format to report to the chat line.
- **Play a Sound?** Select to play a sound clip when the button is activated.
    - **Sound Clip:** The Sound clip file to be played. Select a file in .au, .aiff, .mp3, or .wav format. The sound file specified in this field will be played when the action is invoked.
- **Send Hotkeys?** Select to send hotkeys to other components when the button is activated.

o. **Hotkeys:** The list of Hotkeys to be sent. Use the **New** button to add another key, or the **Remove** buttons to remove existing keys.

To add an Action Button to the Toolbar,

1. Right-click the **[Module]** node, and pick **Add Action Button.**
2. On the **Do Action** dialog, enter the settings for the Action Button.
3. Click **Ok.**

The Action Button component is distinct from the Action Button Piece Trait (see page 43).

Charts Window

A Charts Window is used for displaying gameplay aids, such as charts, tables, and important game information. Charts are accessible using a Toolbar button.

A Charts Window has these attributes:

- **Name**: Name of the **Charts** window.
- **Button Text:** Text for the Notes window button in the Main Controls Toolbar.
- **Tooltip Text:** Mouseover tooltip for the Toolbar button.
- **Button Icon:** Icon for the Toolbar button.
- **Hotkey:** Keyboard shortcut for the Toolbar button.

Sub-Components

A Charts Window is highly configurable, and can contain any combination of tabs, lists, and pull-down menus containing individual Charts, HTML Charts, or Maps. For example, a Scrollable List could include a Tabbed Panel, which includes individual Charts.

Additional Module Components: Game Piece Inventory Window

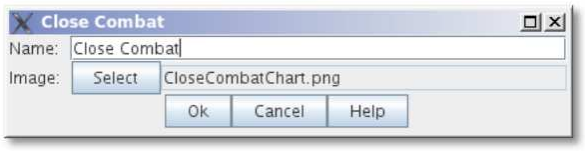
Chart Displays

- **Tabbed Panel:** A panel with tabs, each of which corresponds to a Panel or other Tabbed Pane subcomponent. The label of the tab will be the name of the subcomponent.
- **Panel:** A panel that can contain Charts, HTML Charts, or Maps. Select **Fixed cell size** to specify a fixed number of columns for the panel. Otherwise, the sub-components will appear in a single row, or a single column if the **Vertical layout box** is checked.
- **Pull-down Menu:** A pull-down menu in which each menu item corresponds to a subcomponent. The name of the menu item will be the name of the subcomponent.
- **Scrollable List:** A scroll list in which each entry corresponds to a subcomponent. The name of the entry will be the name of the subcomponent.

Chart Types

- **Chart:** A chart is an image file depicting a game table or other useful information.
- **HTML Chart:** An HTML Chart is a simple HTML page. The HTML should be simple; avoid using the <Head> tag. HTML Charts can contain hyperlinks to one another, and to files in the module, but not to external resources.

- **Map:** A fully functioning Map Window can be embedded within a Chart. Use a Map when you want to place counters onto a chart for bookkeeping purposes.



*You cannot paste a Map Window created as a Chart to the **[Module]** node, or vice versa.*

To create a Charts window,

1. Right-click the **[Module]** node and select **Add Charts**
2. In the **Charts** dialog, specify the settings for the Charts window.
3. Click **Ok**.
4. Right-click the new **[Charts]** node and pick a sub-component to add.
5. Continue adding subcomponents as needed.

Game Piece Inventory Window

A Game Piece Inventory Window organizes and summarizes the pieces in the game in a tree view (similar to browsing a file system). You can define exactly which pieces are displayed in the window and how they are organized.

Possible uses for a Game Piece Inventory (GPI) Window include:

- *Displaying the name and location and location of pieces on a map*: Each unit in an army could be displayed by grid location with other units in its stack. Units in each stack could even be organized in subgroups based on some Property—for example, all Depleted units in the stack could be in a subfolder inside each stack listing.
- *Tracking discarded or □dead□ units*: A hidden map could be created (see page 90), and discarded or destroyed units could be sent there (using the Send to Location Trait) instead of being deleted from the game. Then, a GPI window could list all units sent to the hidden map, which would give an easy to use summary of discarded units without giving access to the pieces themselves.
- *Grouping and listing pieces by some Property:* For example, in a personal combat game, where combatants move in order of their Dexterity, pieces could be assigned a Dexterity property. In the Game Piece Inventory Window, pieces could be grouped by the value of their Dexterity and each group displayed in (ascending) order.
- *A stack management tool:* You can make the Command menus of pieces accessible through the GPI window. Each piece is directly accessible—no unstacking and re-stacking of pieces is required. As a result, for games with large, unwieldy stacks, it□s sometimes easier to use a GPI window to access the individual pieces.

A Game Piece Inventory Window has these attributes.

- **Name:** The name that appears in the window title bar.

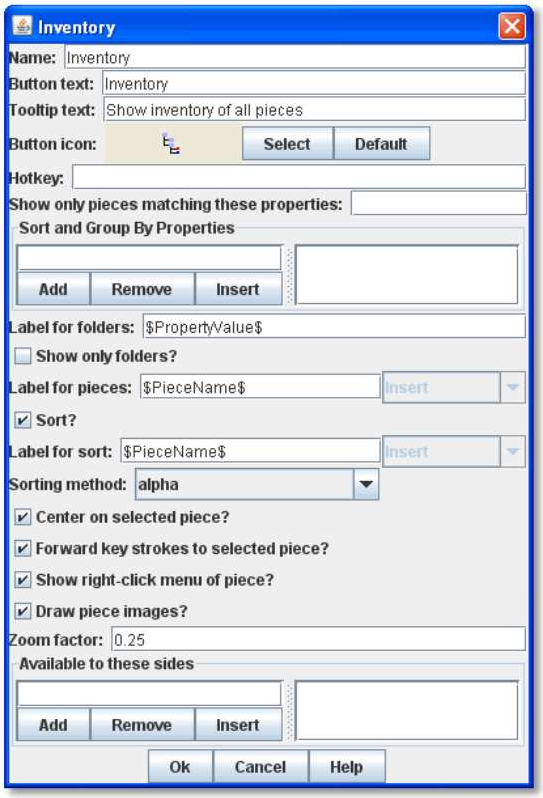
- **Button Text:** Text for the Inventory Window button in the Main Controls Toolbar.
- **Hotkey:** Keyboard shortcut for the Toolbar button.
- **Tooltip Text:** Mouseover tooltip for the Toolbar button.
- **Show Only Pieces Matching These Properties:** The window will only summaries pieces with the matching set of

Properties. For example: limit the pieces to a single map with the CurrentMap Property, or only select pieces with a given value of a Marker Trait.

- **Sort and Group By Properties:** A list of Property names. Pieces with the same value of a given Property will be grouped together at the same level. (Example: listing the

CurrentBoard and LocationName Properties will cause the Inventory Window to show a top-level folder for each board and a sub-folder for each location that contains a Game Piece.)

- **Label for Folders:** A Message Format specifying the text used to label each folder in the tree. The PropertyValue Property gives the value of the Property that defines its group (for example, the board name or location name). Any Property



of the form sum_XXX will be replaced with the sum of Property

XXX for all pieces within that folder. For example, a Game Piece uses a Layer Trait named Manpower, giving it an automatic Property named Manpower_Level. Using the sum_Manpower_Level Property in the folder label will report the total manpower for all pieces inside that folder.

- **Show Only Folders:** If selected, then individual pieces within a folder will not be shown in the view.

- **Label for Pieces:** A Message Format specifying the text used to label each piece in the tree.
- **Sort:** If selected, then sort pieces.

o. **Label for Sort:** A Message Format specifying the text that sorts pieces. (Example: A piece is named *3rd Battalion, 4th Regiment, 3rd Division*; for sorting the markers $division $regiment $battalion are used rather than the piece's name.)

  o. __ **Sorting Method:** Choose a sorting method:

    - *Alpha* sorts the inventory tree alphabetically.
    - *Numeric* sorts by the value of the first integer found, in ascending order. (Descending order is not currently available.)
    - *Length* sorts by the string length first.
    - When two entries are equal for numeric and length, alpha is used for sorting. (Example: id is the Label for sort. Three Game Pieces have the ids 'a', 'aa', and 'b'. Sorting by alpha and numeric is ['a', 'aa', 'b']. Sorting by length is ['a', 'b', 'aa']. Three Game Pieces have the ids 'a3', 'b2', 'c-4'. Sorting by alpha and length is ['a3', 'b2', 'c-4']. Sorting by numeric is ['c-4', 'b2', 'a3'].)
    - **Center On Selected Piece:** If selected, then clicking on a Game Piece in the tree will center the map on that piece.
    - **Forward Key Strokes To Selected Piece:** If selected, then any keystrokes types into the window will be sent as key commands to the selected piece. Selecting a folder will send the command to all pieces within that folder.
    - **Show Right-Click Menu Of Piece:** If selected, then right-clicking on a Game Piece in the tree will display its Command Menu, which can be used to send commands to the piece. (This can be a handy way to manage Game Pieces in large stacks.)
    - **Draw Piece Images:** If selected, the tree will draw reduced-size images of the piece at the specified Zoom factor.
    - **Zoom Factor:** The magnification factor for drawing pieces in the tree.
    - **Available To These Sides:** The Toolbar button will only be visible to the player Sides listed here. An empty list makes the button visible to all players.

To create a Game Piece Inventory Window,

1. Right-click the **[Module]** node, and pick **Add Game Piece Inventory Window**.
2. In the **Inventory** dialog, enter the settings for your **Game Piece Inventory window.**
3. Click **Ok.**

Global Key Command (Module Level)

The Global Key Command (GKC) adds a button to the Main Controls Toolbar. Clicking the button will select certain pieces in the module and apply the same keyboard command to all of them simultaneously.

Global Key Commands are hierarchical. A Global Key command assigned to the module can affect

any pieces in the module.

However, a Global Key command assigned to a map (see page 25) may only affect pieces on that map.

Commands applied by Global Key Commands will be affected by piece ownership. If the GKC triggers a command that is restricted by side, the action may not take place as intended when the restricted side triggers the GKC (by button, hotkey, Turn-based Global Hotkey, or other command).

The Global Key Command has these attributes:

- **Description:** A description of the action, used for the button's mouseover tooltip.
- **Key Command:** The keyboard command that will be applied to the selected pieces.
- **Matching Properties:** The command will apply to all pieces on the map that match the given Property expression.
- **Within a Deck, apply to:** Select how this command applies to pieces that are contained within a Deck.

o. *No pieces* means that all pieces in a Deck ignore the command.

o. *All pieces* means that the command applies to the entire Deck.

  o. _ *Fixed number of pieces* enables you to specify the number of pieces (drawn from the top) that the command will apply to.

    - **Tooltip text:** Mouseover hint text for the Toolbar button.
    - **Button text:** Text for the Toolbar button.
    - **Button Icon:** Icon for the Toolbar button.
    - **Hotkey:** Keyboard shortcut for the Toolbar button.
    - **Suppress Individual Reports:** If selected, then any auto-reporting of the action by individual pieces by the Report Action Trait will be suppressed.
    - **Report Format:** A Message Format that will be echoed to the Chat window when the button is pressed.



Example: Suppose you have configured some pieces to contain a Layer indicating that a Game Piece has fired, activated by Ctrl-F and with the name Fired.

Give each piece the Marker Trait with Property name canFire and value true. Configure the Global Key Command to apply to pieces whose Properties match canFire = true && Fired_Active = true. Specify Ctrl-F as the key command. Now clicking the Global Key Command button will set all marked pieces on the map to not having fired.

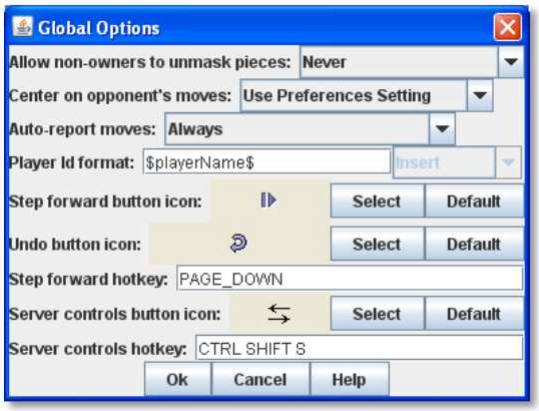
To create a module-level Global Key Command,

1. Right-click the **[Module]** node and pick **Add Global Key Command.**
2. In the **Global Key Command** dialog, enter the settings for the command.
3. Click **Ok**.

Global Options

Global Options are settings that apply to the module as a whole. If an option has a **Use Preferences Setting** choice, selecting it will add an entry **Preferences** window to allow players to choose their own value for the setting at game time.

- **Allow Non-Owners To Unmask Pieces:** By default, only the player who originally masked a Game Piece (see the Mask Trait for Game Pieces) is allowed to unmask it. This option allows other player to unmask a masked piece
- **Center On Opponent's Moves:** This option will center a Map Window in an opponent's move when reading a logfile or receiving a move on the server.
- **Auto-Report Moves:** This option will automatically report a text description (for example, "3rd Cavalry moves from A10 -|B11") to the chat area of the control window whenever a player moves a Game Piece in a Map Window.
- **Player ID Format:** A Message Format that is used to identify players when typing chat text.
- **Icons and Hotkeys:** You can specify your own button icons and keyboard shortcuts for the logfile step/undo buttons and the button that shows/hides the server controls.



Sub-Components

You may add your own arbitrary preference settings to the global options. The different sub-components support different constraints on the values of the preference setting. The values of these preference settings are exposed as Properties.

You must save and re-load the module before these sub-components will show up in the Preferences window

- **String Preference:** A simple string value.
- **Text Box Preference:** A multi-line string value.
- **Drop-down List Preference:** A drop-down from which the user selects from a list of specified values.
- **Whole Number:** An integer value.
- **Decimal Number Preference:** A floating-point value.
- **Checkbox Preference:** A true/false value.

Global Property

Global Properties can be attached to a Zone, Map Window, or Module. The **[Global Properties]** node is a container for all Properties attached to the Map or Module.

When looking for the value of a Property of a Game Piece, global Properties provide default values. If the Property is not defined on the Game Piece itself, the value will come from the Zone occupied the by piece, the Map to which it belongs, or the Module overall, in that order.

A Game Piece can define the value of a Global Property with the Set Global Property Trait. See page 62 for more information.



A Global Property has these attributes:

- **Name:** The name of the Property.
- **Initial Value:** The value of the Property at the start of a new game.
- **Description:** Description of the Property.
- **Is Numeric?** If selected, then changes to the value of the Property will be restricted to integer values.
- **Minimum Value:** Numeric values will be restricted to no less than this number.
- **Maximum Value:** Numeric values will be restricted to no more than this number.
- **Wrap Around:** If selected, then when incrementing this numeric Property, values will wrap around from the maximum to the minimum.

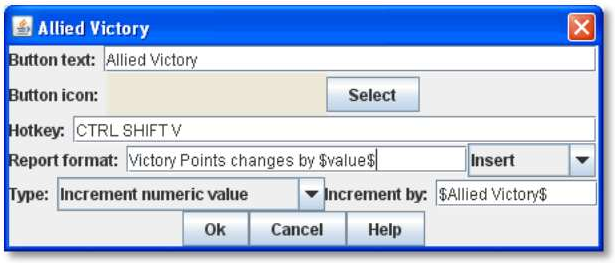
To add a Global Property,

1. Right-click the **[Global Properties]** node, and pick **Add Global Property.**
2. On the Global Property dialog, enter the settings for the Property.
3. Click **Ok.**

Change-Property Toolbar Button

A Change-Property Toolbar button changes the value of a Global Property. Like other Toolbar buttons, you can combine multiple buttons into a single drop-down menu using a Toolbar Menu.

- **Button Text:** The text of the Toolbar button.
- **Button Icon:** The icon of the Toolbar button.
- **Hotkey:** Keyboard shortcut for the Toolbar button.
- **Report Format:** Message Format of a text message to echo to the controls window when the button is pressed: oldValue is the value of the Global Property prior to the button press, newValue is the value after the button press, and description is text from the **Description** field of the Global Property dialog.
- **Type:** Defines how the Property value should change:



o. *Set value directly* sets the Property to a fixed value, after substituting values of Properties.

o. *Increment numeric value* adds a fixed value to the Property. You can specify a number, or the value of another Property. (If you specify a Property, enter the name of the Property in $-signs; for example, $ExampleProperty$.)

o. *Prompt user* displays a dialog for the user to type in a new value.

o. *Prompt user to select from list* displays a dialog with a drop-down menu for the user to select from.

To add a Change-Property button to a Global Property,

1. In the **[Global Properties]** node, select the Global Property to add the button to.
2. Right-click and select **Add Change-Property Toolbar Button.**
3. In the dialog, enter the settings for the button.
4. Click **Ok**. The button is added to the Main Controls Toolbar.

Map Window Toolbars

Each Map Window comes with a Toolbar, which includes button controls for the options you have selected for it. Typically, each of these buttons includes a text label and icon that describes its function. For example, if you have selected additional controls like the Zoom Tool or Line of Sight Thread, the Toolbar for the Map Window will include buttons for these controls.

Main Controls Toolbar

The Main Controls Toolbar is displayed above the main Map Window, at the top of the screen. Every module must have a main Toolbar; it cannot be disabled even if the game does not have a main map window.

The Toolbar comprises these button types:

- **Standard Buttons:** Standard Main Controls Toolbar buttons are common to all modules and are shown on the left – hand portion of the Toolbar. These include **Undo**, **Step Through Log**, **Connect to Server**, and **Retire**. These buttons are configured using Global Options (see page 87).
- **Module-Specific Buttons:** These buttons represent components specific to the module. If a Toolbar button is associated with a component (such as with a Game Piece Palette, Toolbar Menu, or Map Window), the module-specific buttons will appear in the order they appear in the Configuration Window, from top to bottom.
- **Map-Specific Buttons:** If the main Map Window includes any additional map options, their buttons, if any, will be shown on the right-hand portion of the Toolbar.

Keyboard Shortcuts (Hotkeys)

If your cursor is in the Chat Window, pressing a button□s keyboard shortcut when the piece is selected will invoke the corresponding button, just as if the Toolbar button was actually clicked.

Hotkeys can also be invoked by automated commands. For example, a Global Key Command refers to the Hotkey of the command that it applies. In every respect, a Hotkey invoked by automated commands will work the same as if an actual player had pressed the key combination on a keyboard. You can define any unique keyboard shortcut you want as a Hotkey for a particular command. To make it harder to press them accidentally, keyboard shortcuts are usually comprised of more than one key, such as Ctrl-X or Alt-Shift-K.

A keyboard shortcut could be composed of any number of keys pressed at once, but generally use 2 or 3, usually in combination with one of the following keys: Ctrl, Alt/Option, Shift, Meta/Command.

To make them more memorable, when assigning keyboard shortcuts, use key combinations that are reminiscent of the command itself. (For example, Ctrl-R would be an easily remembered shortcut for a Die Roll Button.)

Use these guidelines when assigning keyboard shortcuts.

- Avoid using keyboard shortcuts that players could type inadvertently. For example, a single capital letter M would not be a suitable shortcut, nor would Shift-M, because players could

easily type either in the Chat window during ordinary conversation. However, Ctrl-M or Ctrl-Shift-M would both be suitable.

- Be careful about assigning hotkeys to keys that invoke special functions on your computer. Caps Lock, Backspace, Delete, Home, End, Enter/Return, and so on, are not generally suitable for use as hotkeys. Similarly, the Function (F1-F9) keys at the top of a standard keyboard may serve as hotkeys for various Windows or MacOS functions, and pressing them could cause unexpected operating system functions to be invoked instead of the desired piece command.

Modifying Toolbar Buttons

You can modify Toolbar buttons in a variety of ways.

Setting Toolbar Buttons Icons to Null

Many module components, such as Dice buttons, include a default button icon. By setting a Toolbar button icon to null, you can prevent its default icon from being displayed on the Toolbar button. Only the button text will be shown.

To set a Toolbar button to null, when selecting the button icon, click **Select**, and then click **Cancel**.

If the icon is set to null, make sure you specify some button text, or the button will not show up at all in the game.

Replacing Toolbar Button Text with Icons

By default, Toolbar buttons include a text label, but the text label is actually optional. If desired, you can replace the text label completely with an icon.

Create the button icons first in an image editor. Then, for each control where a button is specified (such as for a Game Piece Palette), in **Button Text**, leave blank, and in button icon, click **Select** and select your button image.

You cannot use this method if the Toolbar button is intended to be included in a Toolbar Menu. You must use a text label for the buttons so the Toolbar Menu can sort them.

*The four standard buttons (**Undo**, **Step Through Log**, **Connect**, and **Retire**) will always appear on the Toolbar even if no text label or icon is assigned to them. If both label and icon are omitted, they will appear as very small, blank, but clickable buttons. To reduce player confusion, always assign a text label, an icon, or both to each of these four buttons.*

Hiding Toolbar Buttons

You can hide Toolbar buttons completely from player view. This is helpful if the hidden buttons are for components that players do not need to access directly, such as for automated Global Key Commands, or to create hidden maps.

To hide a component□s Toolbar button, create a new Toolbar Menu (see page 90) . Leave the button text for the Toolbar Menu button blank. Then, under **Menu Entries**, enter the name of each button you want to hide. (You can add any number of buttons to the hidden Toolbar Menu, so you can repeat this as many times as needed to hide multiple buttons.) Click **Ok.** The buttons are now

hidden in the invisible Toolbar Menu, but will still be accessible to automated game functions.

Modifying Toolbar Button Labels

By enclosing button label text within simple HTML tags, you can use simple HTML format to specify various colors, font weights, and sizes. Example: <html>Bold text<p>with a line break<p>and different colors</html| would display as:

Bold text

with a line break

and different colors

Additional Module Components: Multi-Action Button

Multi-Action Button

The Multi-Action Button combines multiple buttons in a Toolbar into a single button, which replaces the component buttons.

Clicking this button automatically invokes the actions of all the other buttons in the order given (from top to bottom).

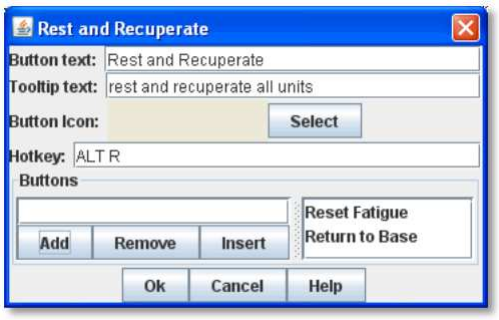
A. Multi-Action Button includes these attributes:
   - **Button Text:** The text of the button to be added to the Toolbar.
   - **Button Icon:** Icon for the Toolbar button.
   - **Hotkey:** Keyboard shortcut for the Toolbar button.
   - **Buttons:** Enter the text of the buttons that you wish to invoke as a result of clicking this button. The text is case-sensitive. They will be invoked in the order listed (top to bottom).

To add a Multi-Action button to the Toolbar,

1. Right-click the **[Module]** node, and pick **Add Multi-Action**

Button.

2. On the **Multi-Action Button** dialog, enter the settings for the Multi-Action Button.
3. Under **Buttons**, enter the name of the first button to be included in the Toolbar Menu, and click **Add**.
4. Repeat Step 3 for each additional Toolbar button.
5. Click **Ok.**



Multi-Action Button Examples

- A Global Key Command is defined that resets the fatigue level of all armies on the map. A second Global Key Command returns them to their home base. A Multi-Action Button can be used to combine both actions into a single button.
- A Dice Button is defined that exposes its result as a Property named Damage. Some Game Pieces are defined with

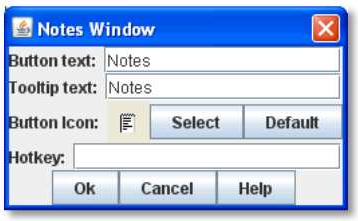
a Trigger Action Trait that compares the level of a Layer (representing armor) with the Damage Property and deletes the piece if the level is below the Damage value. A Global Key Command invokes the Trigger Action. A Multi-Action Button is defined that invokes the Dice Button, followed by the Global Key Command, resulting in the automatic deletion of any units with armor less than the random amount of damage.

- A Symbolic Dice button makes a dice-rolling sound when clicked. The Multi-Action button combines the Symbolic Dice button with a separate Action button, which triggers the dice rolling sound file. For best results, the Action button that plays the sound should be listed first.

Notes Window

The Notes window, accessible by a Toolbar button, enables you to save text notes for a game. The window contains these tabs:

- **Scenario:** Descriptive notes on the scenario. Useful when creating pre-defined setups to describe scenario forces, placement, and victory conditions. Scenario notes are saved when the game is saved.
- **Public:** Notes that are visible to all players, and to which all players may add.
- **Private:** Notes that are visible only to the player who entered them.
- **Delayed:** This tab is for writing messages to be revealed at a later time as a safeguard against cheating. To create a delayed message, click **New** and enter a name and message text. Once created, the text of a message cannot be changed. At the appropriate time, the owning player may reveal the text of the message to other players by selecting the message and clicking **Reveal**.



The number of tabs and their labels are not customizable.

Each **Notes** window has these attributes:

- **Button Text:** Text for the **Notes** window button in the Main Controls Toolbar.
- **Tooltip Text:** Mouseover tooltip for the Toolbar button.

Additional Module Components: Toolbar Menu

- **Button Icon:** Icon for the Toolbar button.
- **Hotkey:** Keyboard shortcut for the Toolbar button.

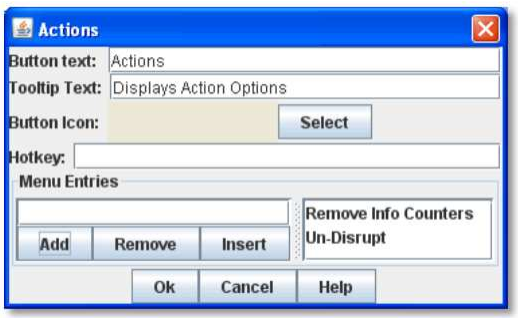
To create a Notes window,

1. Right-click the **[Module]** node and pick **Add Notes Window.**
2. In the **Notes Window** dialog, enter the settings for the Notes Window.
3. Click **Ok**.

Toolbar Menu

The Toolbar Menu component enables you to organize buttons from the Toolbar of the main controls window or a Map Window into a single drop -down menu. Each button named in this component will be removed from the Toolbar and instead appear as a menu item in the drop-down menu. Items added to a Toolbar Menu are case-sensitive.

- **Button Text:** The text of the Toolbar Menu. Clicking the button will reveal the drop-down menu. If left blank, the Toolbar Menu, and any buttons on the menu, will be hidden.
- **Button Icon:** Icon for the Toolbar Menu button.
- **Hotkey:** Keyboard shortcut for revealing the drop-down menu.
- **Menu Entries:** Enter the text of the buttons that you wish to move to the drop-down menu. The menu item will have the same text. If the button uses an icon, the menu item will also use it.



To add a Toolbar Menu,

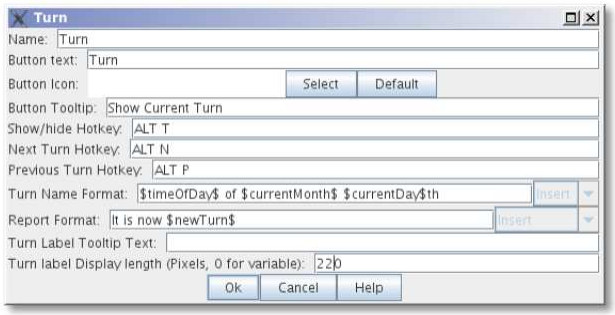
1. Click the **[Module]** node and pick **Add Toolbar Menu.**
2. On the **Toolbar Menu** dialog, enter the settings for the Toolbar Menu.
3. Under **Menu Entries**, enter the name of the first button to be included in the Toolbar Menu, and click **Add**.
4. Repeat Step 3 for each additional Toolbar button.
5. Click **Ok**. The Toolbar Menu is displayed on the Toolbar.

Turn Counter

A Turn Counter can be used to track any intervals you define, such as turns, phases, rounds, segments or days. To mark the progress of the game, players can advance the turn forward or backward, or, optionally, jump directly to a selected turn.

A Turn Counter is defined as a series of nested levels to any level you desire. Advancing the turn moves the deepest level forward. When a child level wraps around, the next child level under the same parent advances forward. When the last child level has wrapped around, the parent level advances forward.

For example, a level representing the Month may contain a level representing the Day, which contains a level representing time of day (Morning or Evening). Advancing the turn counter moves the game from Morning to Evening (deepest level), then to Morning of the next day, evening of the next day, and so on.



Although there is no programmatic limit to the number of nested levels you can use in a Turn Counter, there may be a practical one. Tracking each individual phase, sub- phase and segment of some complex games could mean that the Turn Counter is constantly being clicked to advance the game, which may be a burden during game play.

The Turn Counter controls can be docked into the Main Controls Toolbar, or can be opened in a separate window that is shown or hidden by a button on the Toolbar. Whether the controls are docked is controlled by the player's preferences.

A Turn Counter includes these attributes:

- **Name:** A name for display in the Configuration Window.
- **Button text:** The text of the Toolbar button to show or hide the controls when un-docked

- **Button Icon:** Icon for the Toolbar button.
- **Tooltip Text:** The tooltip text of the button.
- **Show/Hide Hotkey:** Keyboard shortcut to hide or show the Turn Counter window when undocked.
- **Next Turn Hotkey:** Keyboard shortcut to advance the Turn Counter one step.
- **Previous Turn Hotkey:** Keyboard shortcut to return the Turn Counter to the previous step.
- **Turn Name Format:** Message Format to format the display of the current turn. All module-level Properties will be substituted. In particular, the Properties exposed by any child Counters or Lists can be used. In addition, the special Properties level1, level2, etc. can be used to represent the values of the active Counter or List within the Turn Counter. For example: If the Turn Counter contains a Month level, which further, contains a Day level, then level1 gives the value of the Month and level2 gives the Day.
- **Report Format:** Message Format to display a message in the Chat Window whenever the turn changes.
- **Turn Label Tooltip Text:** Tooltip text for the Turn Display.
- **Turn Label Display Length:** Set the number of pixels wide the turn display label should be, or use 0 to let it float to suit the current turn display.

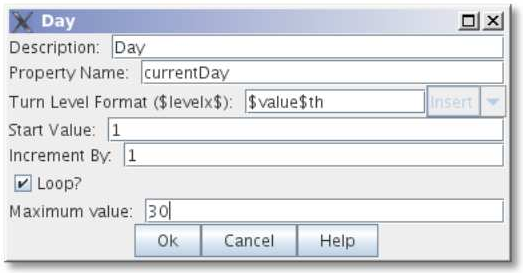
Types of Turns

Turns can be of two types: Counters and Lists. Both types can freely be nested in one another, in any combinations.

Counter

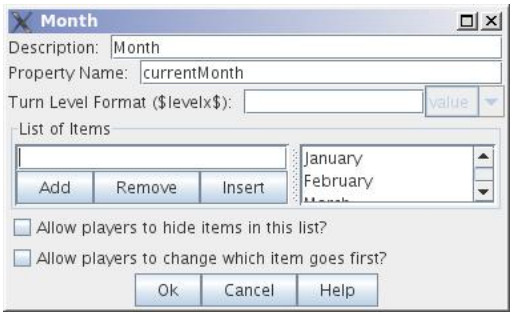
A Counter is a numerical level that advances by incrementing the number by a fixed value. It can optionally loop when it reaches a maximum value. An example of a Counter would be Turn 1, Turn 2, Turn 3, and so on. A Counter has these attributes:

- **Description:** A name for display in the Configuration Window.
- **Property Name:** The name of the global Property that will hold the value of this level.
- **Turn Level Format:** A Message Format that gives the value of the level1, level2, etc. Property for use in the Turn Counter's Turn Name Format Property.
- **Start Value:** The initial (and minimum) numeric value.
- **Increment By:** The amount by which the numeric value increases when the level advances.
- **Loop:** If selected, the level will return to its starting value after reaching the maximum value.
- **Maximum value:** The maximum value at which the level will loop.



List

A List is a level that cycles through a specified list of text strings. An example of a List would be Spring, Summer, Fall, and Winter.



- **Description:** A name for display in the Configuration Window.
- **Property Name:** The name of the global Property that will hold the value of this level.
- **Turn Level Format:** A Message Format that gives the value of the level Property for use in the Turn Counter's Turn Name Format.
- **List of Items:** A list of text strings that the level will cycle through.
- **Allow Players To Hide Items In This List:** If selected, then player will be allowed to disable items in this list at game time.
- **Allow Players To Change Which Item Goes First:** If selected, then players will be allowed to change which should

be the beginning item in the list, i.e. the item at which the parent level will be advanced. Example: If a List represents Sides in a game, but the order in which Sides move is not always fixed.

Additional Module Components: Turn Counter

Turn Counter Properties

Lists and Counters both allow you to define the name of the global Property used to hold the value of the given level, in the **Property Name** entry box.

For example, if you define a Counter that tracks turns numerically, you could enter currentTurn in **Property Name**.

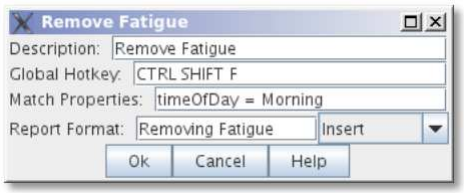
Turn-Based Global Hotkey

A Turn-Based Global Hotkey automatically fires a key sequence whenever a certain state of the

Turn Counter is reached. The Hotkey can trigger the firing of another command or button, such as a Global Key Command, exactly as if a player had typed it in.

For example, when the Repair Units phase is reached, a Turn-Based Global Hotkey fires that corresponds to the keyboard shortcut of a Global Key Command that removes all Damage counters from pieces on the map.

A. Turn-Based Global Hotkey has these attributes:
   - **Description:** A name for display in the Configuration Window.
   - **Global Hotkey:** The keyboard shortcut to fire. The program will respond exactly as if one of the players had typed this key sequence.
   - **Match Properties:** A Property Expression that specifies when to fire the Hotkey. If the expression is true after any level of the Turn Counter advances, the Hotkey will fire.
   - **Report Format:** A Message Format that will be echoed to the Chat Window when the Hotkey fires.



Actions initiated by Turn- Based Global Hotkeys will be affected by piece ownership. If the Turn-Based Global Hotkey triggers a command that is restricted by side, the action may not take place as intended when the restricted side clicks to advance the turn.

For example, Side A in a game represents a group of camouflaged units, which can be hidden (Masked) from Side B at the start of each turn. Each of Side A's pieces includes a Mask trait which only Side A can use. To make things easier, you create a Turn- Based Global Hotkey that triggers a GKC, which causes Side A's pieces to automatically reset their Masks at the beginning of each turn. When Side A clicks to advance the turn, the pieces are masked as intended. However, when Side B clicks to advance the turn, the pieces will not be masked automatically, since Side B Is restricted from using the pieces' Mask trait.

Creating a Turn Counter

To create a Turn Counter,

1. Right-click the **[Module]** node, and pick **Add Turn Counter**.
2. In the **Turn** dialog, enter the values for the Turn Counter.
3. In the Configuration Window, right-click the new **[Turn Counter]** node and do one of the following:
   - Select **Add Counter:** Then, in the **Counter** dialog, enter the settings for the first level Counter.
   - Select **Add List:** Then, in the **List** dialog, enter the settings for the first level List.
4. Optionally, to nest a level under the first one, select either the new **[Counter]** (or new **[List]**)

node, and then repeat Step 3 for the next level.

5. Repeat Step 4 for all further nested levels.
6. Optionally, right-click the **[Turn Counter]** node and pick **Add Global Hotkey**. In the **Global Hotkey dialog**, enter the settings for the Global Hotkey, then click **Ok**.

Tracking Numerical Quantities with a Turn Counter

You can adapt Turn Counters to track a variety of numerical quantities for the game or for individual players. For example, if players in the game must keep track of their Resource Points used to purchase units, you could use a Turn Counter for each player to track Resource Point levels.

In general, to track numerical quantities, you will use a Counter component, and tracker components will not be nested (as they might be with regular Turn Counters).

To create a quantity tracker,

1. Create a Turn Counter.
2. Create a Counter component named for the quantity you wish to track.
3. Set the Start Value of the Counter to the starting level for the game. (If each player began with 40 Resource Points, then you would enter 40.)
4. Choose any other settings required for the Counter.
5. If each player will need such a tracker, copy/paste the newly created tracker as many times as needed to the **[Module]** node, and edit each one appropriately.

Automating an Action to Happen Regularly

Using the Turn-Based Global Hotkey, you can automate a global action to happen on a regular basis, each time the Turn Counter is advanced to a particular level. For example, you have a module where all disabled Infantry units are reset at the end of the Turn, during the End Phase. Since this must occur every turn, automating this will make gameplay faster.

1. Add a command to each unit that will reset its status. Assign this command a keyboard shortcut.
2. Add a Global Key Command to the module.
    a. In **Global Key Command**, enter the keyboard shortcut you assigned in Step 1.
    b. In Hotkey, assign a Hotkey to the GKC. (This is the keyboard shortcut for the GKC itself).
3. On the Turn Counter, add a Turn-Based Global Hotkey.
    c. _ In **Global Hotkey**, enter the Hotkey of the Global Key Command you assigned in Step 2b.
    d. __ For **Match Properties**, enter the turn or phase where the command will be applied. (In the example, this would be Phase = End.)

Now, each time you advance the Turn Counter to the appropriate level, the Global Hotkey will trigger the GKC, which will apply its command to all pieces.

Pre-Defined Setups

Many games include scenarios, where different maps and pieces may be used to simulate diverse game situations. For example, a World War II game could have a scenario for the Battle of the Bulge and another for the Battle of Midway. Each is played using the same module and rules, but would use different maps and counters.

In VASSAL, scenarios are represented by *setups*, which are preset configurations of maps and pieces. Setup files are actually just saved games (.vsav files). You create a setup by setting up the game as appropriate for the scenario, and then saving the file.

VASSAL saves the current game arrangement, including all boards, placement of pieces, and current turn. You can then include the saved game in a module.

If a module includes pre-defined setups, players will be prompted to select a setup when the module is launched. They can also select scenarios from the **File** menu.

About Saved Games

A saved game is a snapshot of the module at the time it was created. This is particularly true of the Game Pieces used in the setup:

- Pieces in a saved game will only include Traits that were part of the piece at the time the saved game was saved.

Changes to a module are *not* reflected in existing Saved Games.

- Decks and At-Start Stacks will only include their contents at the time the game was saved.

If you later modify the Game Pieces to add or remove Traits (in any way, including Prototypes), then the Game Pieces in a setup file created *before* the revision will not be updated to reflect the new Traits. Instead, they will continue to reflect the Traits present on each piece at the time it was created. The same applies of Decks or At-Start Stacks: if you modify them (add, edit, or remove pieces) after the game is saved, the saved game will not show the additions or revisions.

Because setup files reflect the game pieces at the time they were created, creating them should be the *very last task you perform* when creating a module. Only create the setup file when you are *certain* you will not be making any more edits or revisions to pieces or other game components, or you may be forced to create the setup file all over again. (The Saved Game Updater tool can address this issue. See page 97 for more information.)

If you make updates to your module, make sure you also update any Saved Games associated with it. (See I□m Not Seeing My Changes on page 110.)

Creating a Setup File

Create a setup file as a saved game (.vsav) file. Always save the game when logged in as an Observer, to ensure that players can freely select any Side to play. If you do not save the game as an Observer, then when the game is launched, the Side you saved the game as will not be visible for

players to select.

To create a setup file,

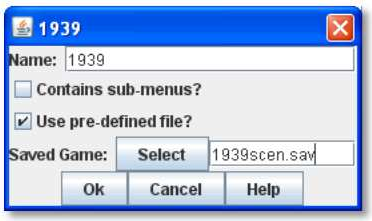
1. Launch the module.
2. Set up the scenario for the game as if you were playing it and take the first player□s side.
3. If the game includes different Sides, click **Retire.** Select another Side to play, then set up pieces for that Side.
4. Repeat Step 3 until all Sides have been set up for game start.
5. Click **Retire** (again).
6. On the **Retire** dialog, click **Become observer**.
7. Optionally, if the module includes a **Notes** window, enter any scenario notes on the **Scenario** tab, and then click **Save**.
8. Click **File** | **Save Game**. Save the game as a .vsav file.
9. The .vsav file you have created can now be added as a Pre-defined Setup, per the below instructions.

The Pre-Defined Setup Menu

The pre-defined setup menu can include two types of item: links to scenario files, and parent menus.

- A scenario is represented by a .vsav file you have included in the module.
- A parent menu is an organizing tool. If you have a number of scenarios, you can group them in one or more parent menus to help organize them for players.

To add a predefined setup or parent menu to a module,



1. Create a setup file as outlined above.
2. Right-click the **[Module]** node and click **Add Pre-defined Setup**.
3. In **Name**, enter the name of the scenario or parent menu.
4. Do one of the following:
   a. If this is a parent menu, select **Parent menu?**
   b. If this is a scenario, select **Use pre-defined file?** Then, click **Select** and browse to the location of the scenario file you created in Step 1.
5. Click **Ok**.

Permitting New Scenarios

If you want to allow players to create their own scenarios at game start, remember to include a blank menu item (named Create New Scenario or something similar). Select **Use pre -defined file**, but do not select any scenario file. Save the menu item. When the players log in, the blank menu item will be displayed with other scenarios. Selecting it will permit the players to create a brand new scenario when the module is launched.

The Saved Game Updater Tool

When a game is saved using one version of a VASSAL module, and then re-opened using a later version, the Game Pieces retain their original behavior, even if the piece has changed in the Game Piece Palette. This is necessary for modules to be backward compatible with old saved games. The **Saved Game Updater** tool enables you to update a game saved with an older version of a module to use the corresponding piece definitions in the current version. The intended use is to save work when creating Pre-Defined Setups for a new module version.

The Updater works by attempting to match each piece in a saved game to the component in the Game Piece Palette or Deck that it came from. The name of the piece in the saved game is matched with a component in the Game Piece Palette.

For example, a Game Piece named "4-6-7" may be defined in a list called "Squads" within a drop-down menu named "German" inside a tab named "Ground Units". The Saved Game Updater notes the component in the Game Piece Palette where the "4-6-7" piece was defined in the old module version, then looks for the same component in the new module version (that is, the "Ground Units" | "German" | "Squads" | "4-6-7" component). If it finds the component, it will replace any "4 -6-7" piece in the saved game with the piece from that component, matching the value of Text Labels, Layer activation, rotation, and other attributes, to the best of its ability.

It is likely you will have to load the updated saved game and make some adjustments to individual pieces, as the process is not perfect.

Because the Updater tool relies on matching piece names with component names in the Module Editor, it will not work well if many different kinds of pieces share the same name, or if the structure of the Game Piece Palette has changed significantly between module versions.

To update a saved game using the Saved Game Updater tool,



1. Back up your saved game files to a separate location.
2. Open the earlier module version in the Module Editor.
3. Select **Tools | Update Saved Games | Export Game Piece** info.

4. Save the info to a file on disk.
5. Close the Module Editor.
6. Open the later module version in the Module Editor.
7. Select **Tools | Update Saved Games |Import Game Piece info**.
8. Select the info file saved in step 3. The module version of saved games field will list the earlier module version number.
9. Click **Choose** and select any number of saved game files in the same folder to update.
10. Click **Update Games** to overwrite the files.

Help Menu

You can supplement your module with a variety of informational files and settings. Help files and Tutorials can be useful in explaining the functions of a complex module to your players.

The **Help** menu in the main control window contains general informational files for your module. You may add more help files specific to the module you are creating. Help files in a module may include information such as game rules, descriptions of how to use particular module features, setup instructions, copyright notices, or other useful text.

Help menu items can be any of these types:

PDF Help File

From VASSAL 3.5 onward you can add a PDF file directly to VASSAL. Selecting the menu opens the default browser on the user's machine with the PDF file you specified. A PDF Help file has these attributres:

- **Menu Command:** The menu item to be added to the Help menu.
- **PDF file:** Click the select button to select a PDF file from your computer. It will be copied into the module and then displayed when the user selects the menu item.

HTML Help File

The HTML Help file component adds an entry to the **Help** menu. Selecting the menu opens the default browser on the user's machine with HTML content that you specify. An HTML Help file has these attributes:

- **Menu Entry:** The menu item to be added to the Help menu.
- **Contents:** A folder on your local file system. The contents of the folder will be copied into the module and expanded onto the user's machine when the user selects the menu item.

The folder can contains any number of sub-folders and can include image data, style sheets, and other associated content. Be sure that any HTML content makes use of relative URLs.

- **Starting Page:** The file within the Contents folder that the user's browser will be pointed to. Normally, this is an HTML page.



o. You can launch a PDF file by setting the **Starting Page** to a local PDF file.

o. You can launch an external URL in the user's browser by specifying the external URL as the Starting Page, and leaving the Contents set to null.

To add an HTML Help file to the module,

1. Create and save the HTML and other files in your choice of file editor (such as an HTML authoring tool).
2. Open your module in the Module Editor.
3. In the Configuration window, right-click the **[Help Menu]** node and pick **Add HTML Help File.**
4. Enter values for **Menu Entry**, **Contents** folder, and **Starting Page**, and then click **Ok**.

Plain Text Help File

The Plain Text Help file component adds an entry to the **Help** menu. Selecting the menu displays a new window with the contents of a plain text file.

The text file can include any text you like. Some examples include:

- A list of design credits and acknowledgements.
- Instructions on how to set up or use the module.
- Rules or rules summaries.
- Reference information.

A Plain Text file has these attributes:

- **Menu Entry:** The menu item to be added to the Help menu.
- **Text File:** A file containing the contents of the window to be displayed. Create and save the text file in your choice of text editor before adding it to the module.

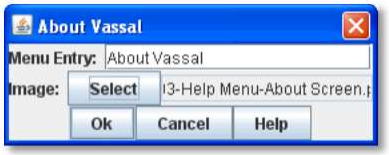
To add a Plain Text help file to the module,

1. Open your module in the Module Editor.
2. In the Configuration window, right-click the **[Help Menu]** node and pick **Add Plain Text Help File.**
3. Enter values for Menu Entry and Text File, and then click **Ok**.

About Screen

The **About** screen will display the currently installed versions of the VASSAL engine and module. This helps players make sure they are running compatible versions.

A new module uses a default image for the **About** screen. However, you may use any image you wish as an **About** screen for your module.



The image for the About Screen is also displayed in the Welcome Wizard when the module is first launched.

The About Screen has two attributes:

- **Menu Entry:** The Help menu item to invoke the screen.
- **Image:** The image displayed when the screen is invoked.

To change the default About screen,

1. Create the new About image in your choice of image editor.
2. In the Configuration window, right-click the **[Help Menu]** node and pick Add About Screen.
3. Enter the values for each setting, and then click **Ok**.

Tutorials

A Tutorial is a short walk-through of your module, played back live by new players. It can include a demonstration game setup, a sample turn sequence, or show the use of module functions.



A Tutorial is really just a logfile that players can step through, reviewing each move in turn, exactly the same as using a VASSAL PBEM log.

Plan your Tutorial to show the most important aspects of gameplay, before adding it to your module.

The **Tutorial** menu item includes these attributes:

- **Menu Command:** The menu item under the Help Menu.
- **Logfile:** The logfile that players will step through when they select the corresponding menu item.
- **Launch Automatically On First Startup:** If selected, then players will automatically be prompted to run the tutorial the first time they load the module.
- **Auto-Launch Confirm Message:** Provides the text in the yes/no dialog that is displayed to the player when they load the module for the first time. Answering Yes will load the tutorial logfile.
- **Welcome Message:** The message that displays in the main controls window chat area when the tutorial is loaded.

To add a tutorial to the module,

1. Launch the module for which you wish to create a tutorial.

2. In the VASSAL player, click **File | Begin Logfile.**
3. Play one or more turns of the game, changing Sides as needed. (You can also add explanatory text in the Chat window.)
4. Click **File | End Logfile.** This ends recording of your logfile.
5. In the Module Editor, in the Configuration Window, right-click on **Help Menu** and choose **Add Tutorial.**
6. In the **Tutorial** dialog, enter the other settings for your tutorial replay, as desired. (In **Logfile**, select the logfile you recorded previously.)
7. Click **Ok**. The tutorial is added to the module□s **Help** menu.

Additional Topics

This section describes some advanced module design topics. It presumes knowledge of the design procedures discussed earlier.

Importing Custom Classes

If you're familiar with Java programming, VASSAL enables you to write and plug your own Java classes into a module. As an open- source project, the VASSAL engine package contains the source code for the core engine. The VASSAL libraries package contains additional .jar files you will need to compile and run the program. The main class for the application is named org.Vassalengine.Main and resides in Vassal.jar.

To import a Java class for a particular component,

1. Right-click the component you wish to import a Java class for.
2. Pick **Add Imported Class.**
3. Enter the name of the class, and click **Ok**.

A complete programming tutorial, which presumes a level of Java programming skills, is available at https://vassalengine.org/wiki/Programming_Tutorial.

Module File Structure

A module (.vmod) file is simply an archive file using ZIP compression. It can be easily decompressed using any application that handles ZIP files, including WinZip or the MacOS Archive Utility, to view or extract the files inside. If you need to have a look at a file inside a module, simply unzip the module.

For some utilities, you may need to change the file extension from .vmod to .zip in order to be able to unzip it. You can change it back to .vmod when the process completes. Make sure that your operating system is set up to display file extensions.

Because a module file is itself a ZIP file, when you packaging or publishing a module, do not zip the module file. For one thing, the savings due to compression is minimal. In addition, compressing a ZIP file can cause confusion when unzipping, as some ZIP utilities may unzip the base ZIP file and any included ZIP files, leaving only the component files behind.

Extension (.vmdx) files are also ZIP files, like module files.

File Components

A module file contains the following components:

- **BuildFile:** The BuildFile is a descriptor file containing all of the module's component settings, in plain text format. You create and modify the BuildFile automatically when you edit a module in the Module Editor.

o. The BuildFile can be opened in any text editor. Each component is specified as a text string, along with values for the component's settings. However, although you can use a text editor to open it, the BuildFile is not particularly legible for humans.

o. Some advanced VASSAL module designers prefer to edit the BuildFile manually, in a text editor. However, the structure of the BuildFile is complex and intricate. *Editing it directly is *not* recommended*. A single typo or misplaced comma can ruin your whole day.

  o. ___ In general, use the Module Editor to make changes to your module; these changes will be automatically reflected in the BuildFile.

    - **ModuleData File:** The ModuleData file contains the module's basic settings. Like the BuildFile, it should not be edited directly.
    - **Images Folder:** Contains all of the module's images: maps, pieces and other images.
    - **Sounds Folder:** Includes any sound files associated with the module (from Play Sound Traits or Action Buttons).
    - **Help Folder:** Contains HTML help files, if any.
    - **Additional Files:** Some modules may contain additional files, such as a Readme.txt file.

Reducing Module File Size

The module design process can lead to module files including unnecessary or obsolete files. Large files are slow to download and unwieldy to distribute. As a result, once the design process is complete, you may wish to reduce the file size of your module.

The vast majority of a module's file size is usually caused by the image files included in the module. You may wish to try the following in order to reduce overall file size:

- Re-scale your game graphics to a smaller size. A large 5000x5000 pixel map may be more manageable when rescaled to 2500x2500 pixels. Use your favorite image editor application to accomplish this.
- Delete unnecessary files from the module, such as unused images or obsolete text files.

Deleting Unnecessary Files from a Module

To reduce file size, you can delete unnecessary files from a module. To accomplish this, you will require a utility capable of unzipping and re-zipping files.

To delete files from a module,

1. Create a backup of your existing module in the same directory as your existing module.
2. Unzip the original module using the unzip utility of your choice.
3. Open the resulting directory. There will be a file called BuildFile, a folder named Images, and some other bits and pieces. (This is the "root level" of your module.)
4. Switch to the Images directory and delete all the obsolete image files.
5. Repeat Step 4 for any other unneeded files, such as text, sound, or help files.

6. Move up one level, to the directory that contains the BuildFile (the □root□ level).
7. Select all these directories and files, and zip them using your zip utility. You should now have a ZIP file in the same directory as the BuildFile.
8. Rename the new ZIP file to the original name of your module: <module name>.vmod.
9. Drag the new ZIP file up one directory level; that is, to the same directory where your backup is. The entry for this module in your Module Manager will now correspond to your new, cleaned-up module.
10. Using the Module Manager, launch the re-zipped module in VASSAL and test it, to make sure you didn't delete any images or other files by mistake. Errors will be displayed in the chat window. Test and check the game thoroughly. If you get an error, use your backup to recover whatever files you deleted and repeat the process from Step 2.

Ensure that you re -zip the module at the root level (this is the level where you see BuildFile, the Images folder, and other files) . Do not zip the folder that contains these items. *If you re-zipped the wrong level, when you open the file in VASSAL, it will return 'Invalid VASSAL module.'*

Importing an Aide de Camp II Module into VASSAL

VASSAL includes a tool that enables you to convert modules made for the application Aide de Camp II (ADC2) into VASSAL module format.

The import process creates maps and pieces and a basic module structure. The tool does not parse any rules or automation in ADC2 modules, so unless the module is extremely simple, the import process will likely require some manual editing in VASSAL after completion to complete the conversion.

To import an ADC2 module,

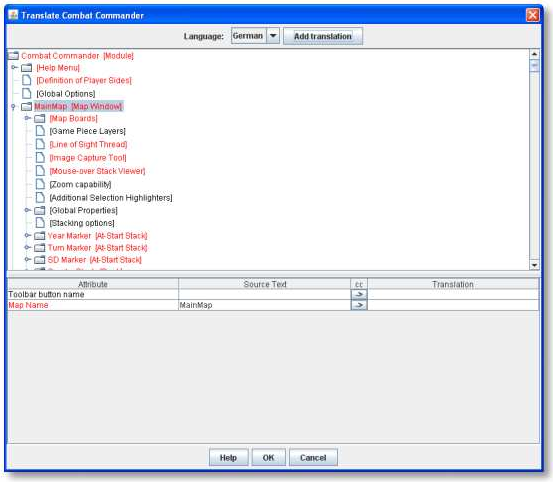
1. In the Module Manager, **pick File | Import Module.**
2. Browse to the ADC2 module you wish to import.
3. Select the .OPS file for the module.
4. Click **Open**. The module is converted into VASSAL format and displayed in the Module Editor.
5. Edit the module as needed and save as a .vmod file.

Translations

VASSAL supports two sets of translations: module translations and translations of the VASSAL engine.

Translating a Module

VASSAL modules are not localized. VASSAL relies on the generosity of module designers (or players) to translate modules into other languages. If you are a fluent speaker of a language other



than English, you can translate the text strings in your module into the language of choice, and save the translated strings. When a player launches the module, VASSAL will use the translation appropriate for the locale of the user's computer.

A module can include translations into multiple languages.

To complete the translation process, first, you specify the language (or languages) into which the module has been translated. Then, you create the actual text strings to be included in the translation to that language.

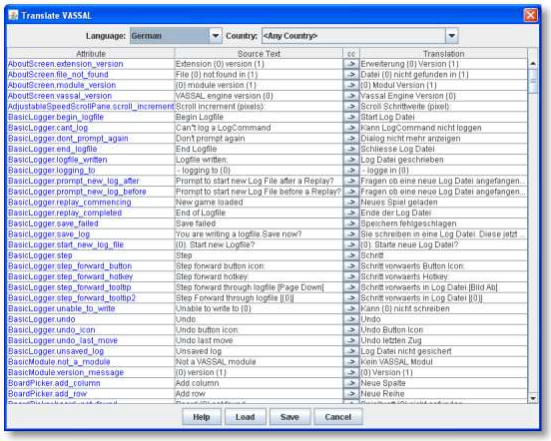
To specify languages for a module,

1. In the Configuration Window, right-click the **[Translations]** node and pick **Add Translation.**
2. In the dialog, in **Language**, select a language from the drop-down list. Optionally, in **Country**, select a country.

To include strings for one of the specified languages,

1. Right-click the **[Module]** node, and pick **Translate**.
2. In the **Translate Module** dialog, in **Language**, select one of the languages from the drop-down list. Any translations you make will be considered to be in this language.
3. In the top pane, module components are shown in a tree view similar to that of the Configuration Window. Module components with text that needs translating are shown in red. Select a component to translate.
4. In the bottom pane, text strings requiring translation are shown in red. Select one.
5. Under **Translation**, double-click the empty box. Then, enter the translation for the selected string into your chosen language.
6. Repeat Step 5 for any other strings.
7. Select a new component to translate from the top pane. Repeat Steps 3-6 for this and any other components.
8. Click **Ok**.

Module translations are not shown in edit mode. The translated strings will only be displayed when the game is played.



Translating the VASSAL Engine

You can also supply translations for the VASSAL engine.

To create a VASSAL translation file,

1. Launch VASSAL from the command line, with the - translate switch.
2. In the **Translate VASSAL** dialog, select the language you are translating into, and optionally, select a country.
3. For each string you wish to translate, under **Translation**, double-click the empty box. Then, enter the translation for the selected string into your chosen language.
4. Click **Save**.

A translation file is saved in the VASSAL home directory. The next time you start VASSAL, it will look in the home directory for a translation file matching your computer□s locale and display the strings.

When your translation file is complete, email it to support@vassalengine.org. It will be bundled with the next VASSAL release for use by other players worldwide!

Creating Module Extensions

An *extension* is a file that adds features or components to a module. For example, for the fictional game “World War II”, the basic module might include 6 maps, each one the scene of an important European battle. Later, the “North Africa” extension adds 2 more maps that show the scenes of two important North African desert battles, as well as counters representing new tank models. When the extension is loaded, players can select from the older maps or the newer maps, and use the new counters as well.

A module can include any number of extensions.

You should be familiar with creating and editing modules before attempting to create or edit an extension.

What an Extension Can Do

An extension can add functionality or components to a module. Here are some examples of what an extension could add to a game:

- New Game Pieces (units, cards, or any other pieces).
- New Cards to a Deck.
- New maps or boards.
- New tools, such as die rollers or charts.
- New Prototype Definitions (see Extension Prototypes in the Base Module, below).

What an Extension Can□t Do

An extension cannot do any of the following:

- Add new Traits to Game Pieces already defined in the base module.
- Remove, disable, or modify existing components from the base module (such as boards, buttons, and other controls).
- Modify, replace, or override Prototype Definitions from the base module.
- Add new Sides.

Using the Extension Editor

An extension is created using a special editor called the Extension Editor, which is similar to the Module Editor.

In the Extension Editor, the process of adding components is the same for extensions as it is for modules: right-click on a node of the tree, select the component you wish to add, and specify its settings. Note that in the Extension Editor, the components of the main module are displayed, but are disabled, indicating that they may not be modified or deleted by the Extension Editor.

Extensions, and components defined in them can refer to global items defined in the base module, such as Prototypes, Global Properties, Global Key Commands, and Global Hotkeys. For example, if the base module included a Prototype definition called *Zombie*, pieces created in the extension could be assigned the Zombie Prototype. (This is a two-way street. See *Extension Prototypes in the Base Module* on page 107 for more information.)

Extensions are given the extension .vmdx. Like modules, they are ZIP files and can be decompressed using any ZIP-capable utility.

Extension Properties

Each extension has the following Properties.

- **Version Number:** Revision number of the extension.
- **Description:** Brief description or title of the extension. This will be displayed in the Module Manager.
- **Extension ID:** An extension ID links counters in existing save games to the counter definitions in the extension. If this ID is changed, then the Saved Game Updater may not be able to update the counters from existing save games.

Universal Extensions

Although extensions are usually intended to enhance a specific module, it is possible to create a universal extension that can be used by any module.

For example, you could create a universal extension named Percentile_Dice includes a die roller which randomly generates a number from 1-100. You could then use this extension for any module that requires random numbers be generated in such a range.

To create an extension,

1. In Module Manager, right-click on the module for which you wish to create an extension. Then, select **New Extension.** The Extension Editor opens.
2. On the Extension Editor Toolbar, click **Extension Properties.**
3. In the **Extension Properties** dialog**,** do the following:
    a. Enter a version number and description of the extension.
    b. __ If desired, enter an Extension ID.
    c. __ If you wish this extension to be used with any module, select **Allow loading with any module.**
    d. Click **Save** to save the Properties.
4. In the Extension Editor, add or edit components as desired.
5. On the Extension Editor Toolbar, click **Save**. Save your extension with the suffix .vmdx.

To edit an extension,

1. In Module Manager, select the extension you wish to edit. Right-click and pick **Edit Extension**. The Extension Editor opens.
2. Add or edit components to the extension as desired.
3. On the Extension Editor Toolbar, click **Save**. Save your extension with the suffix .vmdx.

Copying, Pasting and Editing Module Components

One useful technique for adding new components to an extension is to copy and paste a similar component from the base module into the corresponding node of the Extension Editor, and then edit the copy as required. Although you cannot change the base module, you can utilize its components in the Extension Editor. (The same strictures apply for pasting as they do for the Module Editor; you can only paste like to like. For example, you could paste Game Pieces from the module palette into an extension palette, but you could not paste a Game Piece copied from the module to a turn counter in the extension.)

Integrating Extensions

Although an extension cannot itself modify the components in the base module, it's sometimes useful to manually modify the base module, in the Module Editor, to make room for extension functionality and improve integration.

Extension Toolbar Menu Items

An extension may add new buttons to the base module's Toolbar. Ordinarily, these new buttons would be displayed separately from the module Toolbar buttons. However, if want to add them to an existing module Toolbar menu, simply add the names of these buttons to the Toolbar Menu component of the main module, in the Module Editor.

For example, the main module includes a Toolbar Menu named Maps that includes the game's 3 basic maps: Map 1, Map 2 and Map 3. You later create an extension with Map 4. In the base module, in the Module Editor, you would add *Map 4* to the Toolbar Menu component. Now the Toolbar Menu would include all of the game's maps (1 through 4). However, the Map 4 menu item would not be displayed until the Map 4 extension is loaded.

Extension Prototypes in the Base Module

Prototypes in the base module are usable by pieces in the extension, and Prototypes defined in an extension will be available to pieces in the base module when the extension is loaded. This enables you to add optional functionality to the base module, which would be activated by loading an extension, and requires that you edit the base module.

For example, we add a Prototype trait called Extension1 to every counter in a base module. However, no Prototype named Extension1 is defined in the base module. When a user uses the base module with no extensions loaded, the Prototype Trait Extension1 is ignored because the definition does not exist in the module, and it has no effect on game play.

We then create a Prototype Definition named Extension1 in an extension with the appropriate

Traits. When the base module is used with the extension, all counters defined in the base module will now have the extended Traits defined in the Extension1 Prototype.

Testing Your Extension

If the Extension Editor is open, launching new games will launch the base module with the extension loaded (automatically activated), enabling you to test the extension like you would a module.

You can only test one extension at a time this way. To test multiple extensions together, you will need to close the Module Editor and the Extension Editor, and launch the game from the Module Manager into regular play mode.

Activating an Extension

In order for a player use an extension, it must be *activated*. For information on activating extensions, consult the *VASSAL User□s Guide.*

Example: Creating an Extension for a Card Game

Card-based games often include expansion sets that increase the number and variety of cards available for play. Creating an extension for such expansion sets is straightforward, particularly if the extension requires no new rules or game functionality.

You should be familiar with working with the Extension Editor, before attempting to create an extension for a card game.

Scan, create, or otherwise acquire all of the graphic images for your new cards before beginning.

1. Open the base module in the Extension Editor.
2. In the Extension Editor, locate the card deck **([Deck]** node) you wish to add cards to. (It will appear disabled and grayed-out).
3. Expand the view of the **[Deck]** node to display the cards in the deck.
4. Right-click a sample card in the deck and pick **Copy**.
5. Right-click the **[Deck]** node and pick **Paste**. You will now be able to edit the pasted card to reflect a card from the expansion. You can change the card name or basic image, add new Traits or Prototypes, or otherwise edit the new card as needed.
6. Repeat Steps 4-5 for any remaining new Cards from the expansion.
7. On the Extension Editor Toolbar, click **Save**. Save your extension with the suffix .vmdx.

You can now test and activate your extension.

To add complexity or new functionality, your extension could include new Prototypes to reflect new types of cards available in the extension.

Publishing Your Module

Once your module is complete, you can publish it to the vassalengine.org web site, for sharing with other VASSAL users.

There are over a thousand VASSAL modules already published and more are available all the time.

- *If the module is brand new,* and no other module exists for the game, you can create a new module page on vassalengine.org for the module and any related files. Look up the game's page on Boardgamegeek.com. The name listed on Boardgamegeek must be used as the title of your new page.
- *If there is already a module for your game*, no worries. You can still upload it. There can be any number of modules for the same game—the more, the merrier! You will not need to create a new page, however; simply edit the existing page to accommodate your new files.

o. In the **Files** section of the page, add new links to your newly uploaded files (module and extensions, if any). Make sure any new files you add have different names from the existing ones, or you will overwrite someone else's files.

o. In the **Comments** section, describe your new module.

For instructions on how to create a new module page, or edit an existing one, see

https://vassalengine.org/wiki/How_to_Create_a_Module_Page

File Types

The following file types can be uploaded to vassalengine.org:

- .vmod for modules
- .vmdx for extensions (.vext is also permissible)
- .vsav for saved game files
- .vlog for log files

Do not place VASSAL files into ZIP file archives. Use the proper registered extension type listed above.

Limitations

Because of copyright issues, these publishers have specifically asked that modules for their games not be uploaded to

vassalengine.org:

- Games Workshop
- Avalanche Press

- Sabretooth Games
- SPI (Decision Games) modules ***may*** be uploaded ***except*** for those related to the following specific games/series: *D-Day at Omaha Beach series, RAF series, War in Europe, and Struggle for the Galactic Empire*.

If uploaded, such modules will be removed from the site, and designers who upload them risk having publication rights to the site revoked.

Other publishers (and copyright holders) may also ask that specific modules be removed. Vassalengine.org always complies with such requests.

Module designers who take exception to the stated copyright policies of these publishers are urged to discuss their concerns with the publishers.

More Information

For more information on publishing your files, see https://vassalengine.org/wiki/Module_Section_Information

Updating a Module

Inevitably, your module will need to be updated. A new feature or functionality could be added to the VASSAL engine; you may get a better idea on how to implement one of your module□s features; or the game itself may be produced in a new edition.

To update a module,

1. In the Module Manager, right-click the module and pick **Edit Module.**
2. In the Configuration Window, right-click the **[Module]** node and pick **Properties**.
3. In **Version Number**, enter a version number higher than the existing one. Click **Ok**.
4. In the Module Editor, make your edits as needed.
5. Click **File | Save As**. Save the updated module with a new filename to reflect the new version number.

Update Guidelines

Follow these guidelines when updating your module.

- **Latest Version:** Make sure you have the latest version of the VASSAL engine installed when updating your module.
- **Version Numbering:** The version number of a module update must be higher than the one used in an earlier version. Some designers like to use the major/minor version numbers (x.y) common in the programming world. A change to the major version number reflects large changes to the module; a change to minor version reflects smaller ones. For example, a change from 1.0 to 2.0 reflects a major revision (perhaps a new board or toolbar), while 1.0 to 1.1 reflects a lesser update (such as fixing a typo). You can revise the version number in the **Properties** of the **[Module]** node.
- **Filename:** Always include the new module version number in the updated filename, so players can quickly tell which version they have without opening it. For example, chess_2.6.vmod would indicate version 2.6 of a chess module. (Use the **Save As** button to save the module with a new filename.)
- **Graphics:** When updating graphics, if possible, use the same filenames as graphics that exist in the module already. This way, the new graphics will *replace* the old ones in the module file, rather than adding new, unnecessary files, which can cause the module to become bloated.
- **Saved Games:** Remember to update any saved games (Pre-Defined Setups) that the module includes. *See I□m Not Seeing My Changes*, below, for more explanation. Remember to add the updated saved games to the module.
- **Update Extensions First!** Making major changes to the structure of a base module can cause issues with editing any extensions associated with the module, particularly when the extension depends on components from the base module.

o. For example, you revise a Game Piece Palette in the base module from a Tabbed Panel to Scrolling List. However, there are pieces in an extension that are assigned to the Tabbed Panel, which no longer exists. In this case, the extension would fail to open because it would depend

on the structure of the base module, which has changed.

o. If you plan to make revisions to a module that includes extensions, always edit the extensions first before editing the base module. In the example above, you might create a temporary palette in the Extension that would correspond to the new structure of the base module, and assign the pieces to it. Then you could edit the base module with the new palette. Finally, you could re-edit the extension and assign these pieces to the new palette.

I□m Not Seeing My Changes

If you don□t see your edits reflected in the game, this may be because you are actually looking at a saved game, not the actual module. If your module includes saved games (Pre-Defined Setups), then after making updates, you will not see those changes take place in the module until you update the saved games. (This is a common oversight among module designers when updating modules. See *Pre-Defined Setups* on page 97 for more information.)

To rectify this, launch your module and load the saved game. Make your updates to the saved game, adding any newly defined pieces or other components, and save it again. Then re-add the updated saved games to the module. (You may need to use the Saved Game Updater to make sure that any Game Piece updates are reflected in the new saved game.)

Extensions and Changing Filenames

Extensions for a module are located in the directory <Module File Name>_ext. For example, extensions for version 1.0 of the *Global Thermonuclear War* module, named gtw_1.0.vmod, are located in the directory gtw_1.0_ext.

If you change the file name when updating and include the new version number in the filename (such as to gtw__2.0.vmod), then Module Manager will no longer be able to locate the extension files from the old version of the module.

As a result, the extensions will not load automatically with the new version. This is easy to fix, but players will need to repair this individually in their own Module Manager.

To re-add existing extensions to the updated module,

1. In Module Manager, locate the new version of the module. (Check the **Module Version** column for version number.)
2. Right-click the new version and pick **Add Extension**.
3. Browse to, and then select, an existing extension file.
4. Repeat Steps 2-3 for any other extensions.

VASSAL will re-locate the existing extension files from the old directory <OldModuleName>_ext to a new directory named <NewModuleName>_ext. (The new extension will be created on the player□s system automatically.)

Each extension will be activated by default. For more information on activating and deactivating extensions, see the *VASSAL User□s Guide*.

New Versions of Existing Modules

As an open-source project, VASSAL modules are freely editable by others. If you've got an idea to improve an existing module, either for your own use or that of others, have at it! You may want to contact the module designer and propose a collaborative effort, or you may wish to use the existing module as a starting point for your own new version. There can be any number of VASSAL modules for a given game.

It's always courteous to inform the original module designer or maintainer of your efforts. Contact email addresses of module contributors are maintained on each module's page.

Best Practices

VASSAL is a powerful toolkit. As a result, there may be any number of ways to accomplish a particular goal, all of which may be equally correct. It's not possible in this guide to outline all the methods for accomplish a given objective.

There are, however, a set of best practices gleaned from other VASSAL designers that can save a lot of time and effort.

The following guidelines may be helpful to keep in mind when designing a module or extension.

Know the Game

Get to know the game before creating a module. You should review the rules in detail even if you've played the game itself many times, looking for potential issues that can slow down the module design process. How does the game proceed? Are there any special rules that standard VASSAL tools can't handle and may require custom coding?

Don't Try to Enforce the Rules

VASSAL is not, in general, intended to enforce rules. Rules enforcement should be left to the players, just as it would be at a tabletop. Is it possible to make a module track each piece's movement points, and types of attacks, and attack results? Probably. Is it a good idea? No. The process will probably involve much work on your part, for little advantage. In addition, VASSAL may not include the tools you need for proper rules enforcement. A tabletop game proceeds by player cooperation and mutual knowledge of the game rules; take your cue from that.

Have All the Required Files on Hand

It's a good idea to collect all the art, text, and other files you're going to use in the module *before* embarking on the creation of a module. The creation process will go much more smoothly if you have all the files you'll need ahead of time. The art can be gathered by scanning the original artwork, you can create your own from scratch, or use some combination of the two. You can create text and help files in any text or HTML editor.

Carefully Target Automation

There's no denying that automation can be a powerful tool for making gameplay easier. Global Key Commands and Trigger Actions are excellent methods of automating nearly any standard game function. Bear in mind, however, that VASSAL is intended to mimic the experience of sitting at a table with your opponents. As a result, the less automation in the game, the better. During game play, too much automation reduces the flexibility players have, as decisions may be made for them automatically.

In addition, a highly automated scheme can make it harder to track down logical errors and game misfires during the module development stage. Another issue could be the computational time

required for automated tasks to execute. In most cases, automation is best used for bookkeeping or repetitive tasks.

Programmatic Efficiencies

Gameplay can be made more efficient and less time-consuming by taking advantage of VASSAL's features. For example, many board games include a turn track, which is an actual physical space on the game board where a turn tracker piece is moved. Use of VASSAL's turn counter can easily replace this, saving you the trouble of creating the turn track window and graphics. Status markers that require placement on top of other counters can be simulated by the use of the Layer trait, without having to drag markers to the board.

Documentation

Players always welcome documentation on how to use your module. Although they may be well-versed in the boardgame version of the game you've designed, a VASSAL module may present unique challenges when it comes to gameplay that need additional explanation. What may be obvious to *you* may not be so obvious to players.

You can use the **Help** menu to add your own designer' s notes or getting started guides, explaining module concepts. If the game or module is particularly complex, consider adding a Tutorial to show how the module is intended to work.

Best Practices: New Versions of Existing Modules

Play a Game!

Test out your module before you publish it by playing some full games. You may realize you've forgotten about a rule or consideration that may only come to light through gameplay. Professional game designers insist on rigorous playtesting, and you should too.

Learn from Others

There's no need to reinvent the wheel. With nearly one thousand VASSAL modules available, chances are high that a game similar to yours has already been turned into a module. In fact, most module designers got started by examining the workings of existing modules and learning from them. Previous module designers may already have solved some of the issues that your game presents. With modules easily available for view in the Module Editor, why not examine previously used methods and learn from them?

The vassalengine.org site also features a forum for asking module design questions.

Tutorials

Two tutorials are presented here, putting the module design process into practice.

Board Game

This tutorial explains the design of a module for a fictitious game called *Zap Wars* , which depicts the epic struggle between a planet of heroic Fuzzy Creatures and their archenemies, the Flesh-Eating Zombies.

The imaginary *Zap Wars* boxed set includes the following components:

- A strategic map for strategic movement between the Fuzzy and Zombie planets.
- A tactical map, where counters are moved to resolve battles. After the battle, counters are moved back to the strategic map. The tactical map is marked in a rectangular Grid to regulate movement. Units on the tactical map can maneuver and change their facing on the tactical map to any of the 4 sides of a Grid cell.
- A Tension Track, a space on the board where dramatic tension levels for each Side are recorded.
- Each Side can deploy an unlimited number of units (spacecraft and related forces).
- The Zombie units can enter an Undead state, which gives them some additional powers. In particular, all units in the Undead state can fire the dreaded Undeath Ray.
- The Zombie arsenal includes a unit called the Minefield, which can be placed in a hidden location on the map.
- Battles are resolved using 2 ten-sided dice (d10s).

All data for this tutorial can be found on the vassalengine.org wiki in the File:Zapwars.zip file.

Getting Started

Launch VASSAL. In the Module Manager, pick **File | New Module.** A new, empty module is created, shown here.

We right-click the **[Module]** node, and in the resulting dialog, enter the following:

- **Game Name:** *Zap Wars*
- **Version Number:** *1.0*
- **Description:** *Invasion of the Flesh-Eating Zombies!*

We can now begin creating the module.

Sides

Sides are usually optional when creating a module. However, in *Zap Wars*, it will be required: since some Zombie units will be hidden from the Fuzzy player (using the Invisible Trait), we want to ensure that the module can clearly distinguish between Sides.

- Double-click **Definition of Player Sides.**
- Enter *Fuzzy* and click **Add**, then enter *Zombies* and click **Add**.
- Click **Ok**.

Now when the game begins, each player will be prompted to select one of these Sides to play as.

Maps, Boards, and Grids

The two maps in *Zap Wars* will each be represented by a different Map Window.

The Tactical Display

The new Zap Wars module comes with one Map Window by default, and we will make this the Tactical map.

Map Window

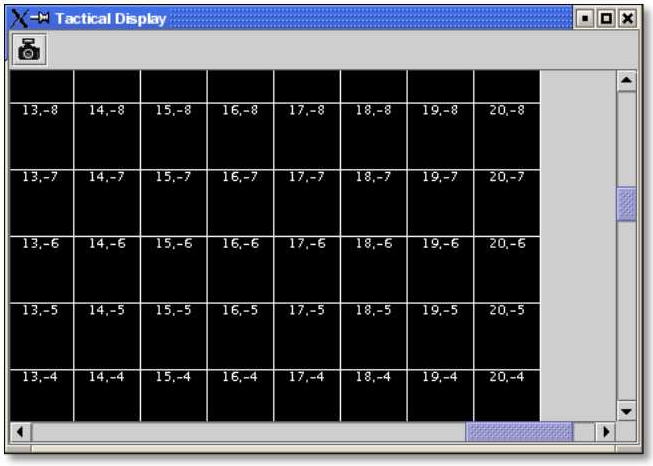
In the Configuration Window, double-click the **[Map Window]** node.

- Give the map a horizontal and vertical padding of 150 pixels each. This gives the window some blank space around the Grid where players can line up their reinforcing ships before they join the battle.
- The map will have a solid black board, so we should choose a different border color for highlighting selected pieces. Next to **Border for selected counters**, click the color selector and choose a bright green color.
- The Tactical map isn't the main playing area. It's only needed when the strategic situation dictates a battle. So we'll check the **Include Toolbar button to show/hide option**. Type *Tactical* for the **Toolbar Button Name**. Place your cursor in the **Hotkey** field and press (together) Ctrl-SHIFT-T on your keyboard. Now notice that the main control window has a **Tactical** button. When you start a game, the button will become enabled and pressing Ctrl-SHIFT-T will bring up the Tactical window.

Board: A Map Window requires one of more Boards, so we need to create a Board for our new window. Open the new **Tactical Display [Map Window]** component.

- Right-click on the **[Map Boards]** node and select **Add Board**. Name the board **Tactical Grid.**
- We'll want the Grid to be 41x41 square with each square being 50 pixels on a Side. We need one extra pixel to draw the complete Grid, so choose 2051x2051 as the size. (We will add the actual Grid shortly.)
- In **Background color**, click the color selector and set the background color to black.

Grid: *Zap Wars* needs a Grid to regulate movement. Rather than constructing a map cell-by-cell, VASSAL defines a complete board and then imposes a Grid on top of it.



Expand the **[Map Boards]** node, right-click on the Tactical Grid component and select **Add Rectangular Grid**. Choose 50 for the width/height and 25 for the x/y offset. Check the **Show Grid** box; then click the color selector and pick white.

Finally, we can assign a numbering scheme to the Grid. Right- click on the **[Rectangular Grid]** component and select **Add Grid Numbering**. The numbering dialog gives you many options for assigning a numbering scheme to the Grid. The numbering scheme is used when reporting the movement of units, but it can also be drawn directly on the Grid. In *Zap Wars,* the tactical Grid cells are numbered x,y with 0,0 in the center. We choose ',' for the separator, -20 for the horizontal/vertical starting number, 0 leading zeros, and Numerical (as opposed to Alphabetic) numbering.

Check the **Draw Numbering** box and select white for the color.

Congratulations! You've defined your first VASSAL Map Window.

Now select **File | New Game** in the controls window. The **Tactical** button becomes highlighted, and clicking it will show the Tactical Display window. The window has a Toolbar with the Image Capture tool. Clicking this would let you capture the entire map to a graphics file in PNG format. (A simple screen capture wouldn't do, since the map is probably too big to fit entirely on your screen.)

It would be a good idea to save your module at this point before continuing. On the Configuration Window Toolbar, click **Save**, and save your module as *zapwars_1.0.vmod.*

The Strategic Display

The main playing area for *Zap Wars* is the strategic map, plus a Tension track. For both of these, we'll use pre-defined artwork. The ZapWarsData folder contains a Strategic.gif and a TensionTrack file.

In the Configuration Window, right-click on the **Zap Wars [Module]** node and select **Add Map Window.**

Map Window

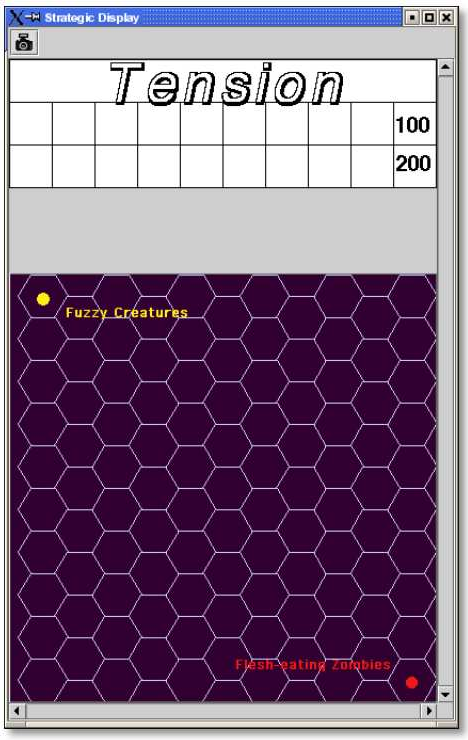
Name the window *Strategic Display*. As you did earlier, set the border highlight color to green. This time, we'll leave the **Include Toolbar** button unchecked. This will cause the Strategic Display

window to always be visible during a game. We'll also check the **Can contain multiple boards** box.

Boards

The Strategic map and Tension Track will each be a separate board that is combined in the window.

- Expand the Strategic Display Map Window node, right-click on the Map Boards component, and select **Add Board**.
- For Board Name, enter *Strategic Map.*
- For board image, click **Select** and select the Strategic.gif file.
- Repeat the process for the second board and the TensionTrack.gif file.



Grids: The Strategic and Tension Track boards have map Grids included in their artwork. We will still add Grids to them to regulate placement of units, but the VASSAL-imposed Grid will be invisible.

- The Strategic board Grid takes a hex Grid with x offset 33, y offset 22, hex height 40.
- The Tension Track takes a rectangular Grid with x/y offset 20 and width/height 40.

In practice, you'll want to follow the guidelines for aligning a Grid given on page 35.

Board Placement: In the Strategic Display Map Window, the Tension Track should go above the Strategic Map.

- Double-click on the **[Map Boards]** component of the Strategic Display component and click **Select Default Board Setup.** A dialog is presented for arranging the boards in the window.
- Click **Add Row** to place two boards on top of one another. In the top slot, select the Tension Track board from the drop-down menu, and select the Strategic board in the second slot.

That completes the definition of the maps in our Zap Wars module. During play, players will drag pieces from the Strategic display to the Tactical display to complete their battles, then drag them back to the Strategic display when finished.

Counters

We need a way to generate Game Pieces for the game, so we will a Game Piece Palette. (You'll find artwork for the counters in the ZapWarsData folder.)

Game Piece Palette Structure: By default, each module is configured with a single Game Piece Palette. First, we'll define its basic structure of the Game Piece Palette. We'll create two tabs: one for each Side, the Fuzzy Creatures and the Flesh-Eating Zombies. The Fuzzies tab will have two different pieces while the Zombies tab will have a scrollable list of different pieces.

- Double-click on the **[Game Piece Palette]** component and enter *Zed Warriors* for **Name** and for **Button Text.** This will be the name of the window containing the pieces. Enter
- Right-click on the Zap Warriors Palette and select **Add Tabbed Pane**l. For Name, enter *Counters*.
- Right-click on the new Counters Tabbed Panel component and select **Add Panel.**
- Set the **Name** to *Fuzzies* and the **Number of Columns** to *2*.
- Right-click again on the Tabbed Panel and select **Add Scrollable List**. Name the list *Zombies*.



Click the **Zed Warriors** button in the Main Controls Toolbar to see the new palette window. We can now add Game Pieces to the Palette.

Basic Piece

The simplest possible Game Piece in VASSAL consists of a single image. We need to create a unit called Fuzzy Base.

- Right-click on the **Fuzzies [Panel]** component and select **Add Single Piece**. You'll be presented with the Properties dialog for adding Traits to a Game Piece.
- Double-click on *Basic Piece* in the **Current Traits** list on the right.
- Set the **Name** to *Base*.
- Double-click on the indicated area on the left Side of the dialog, and select FuzzyBase.gif from the tutorial directory. Now click **Ok**. You'll see the new piece appear in the Fuzzies tab. (The FuzzyBase.gif image uses transparency to give it a shape other than a square.)

Traits

You can customize the behavior of your pieces by selecting Traits for them.

Delete: We should to add the Delete Trait to the new Fuzzy Base unit, or counters won't be able to be deleted from the game after creation.

In the **Fuzzies [Panel]** node, double-click the new Fuzzy Base piece. In the **Available Traits** list, pick *Delete* and click **Add**. The Trait is defined with a default name and default keyboard shortcut. Click **Ok**.

Rotation: One of the most common Traits is the ability to rotate. In the Fuzzy counter mix, bases can't rotate, but warships can. We will create a Warship piece to use this Trait now.

- Right-click on the **Fuzzies [Panel]** node and select **Add Single Piece** again. Set the name to *Warship*. Select FuzzyShip.gif as the base image and click **Ok**.
- Now, from the **Available Traits** list, select Can Rotate and click **Add** to add the Trait to the **Current Traits** list.
- The Can Rotate dialog is now shown. For **Number of Allowable Facings,** enter 4 (which will enable each Warship to rotate up, down, left or right.)

We also add Delete to the Fuzzy Warship as we did for the base.

- You can test your counters without having to drag them onto a map. In the main piece definition dialog, you can right-click on the counter at the top of the window to bring up the piece's popup menu, or select the piece and type. You can do the same with the piece in the Game Piece Palette. When you select the Fuzzy warship and type Ctrl-] and Ctrl-[, the piece will rotate clockwise and counterclockwise.

This completes the creation of the Fuzzy units. Now we want to create the Zombie base and Zombie Minefield.

Layers: Layers are the most common way of adding functionality to a Game Piece. A Layer is a set of images drawn on top of the basic piece. The user can toggle the images on and off, and cycle through them with key commands.

The Zombie base has two states: normal and Undead.

- Right-click on the **Zombies [Scrollable List]** component and select **Add Single Piece**.
- For **Name**, enter *Zombie Base*, but do not select an image.
- Select Layer from the Available Traits and click **Add**.
- Each image that can be cycled through in a Layer is called a Level. We need two levels: one for each state. One of the two levels will always be drawn, so select **Always active**.
- Pick ZombieBase.gif for Image 1, and then click the **Add Level** button.
- Select ZombieBaseUndead.gif for Image 2.
- The **Increase/Decrease** commands are what the players use to cycle through the levels. Since there are only two levels, we don't need both commands. Change the **Increase** command to *Undead* and the key to Ctrl-U. Now when players select a Zombie base and click Ctrl-U, the base will toggle between its normal and Undead states. If we set the name of level 2 to *Undead* and check the **is prefix** button, then when the Undead level is activated, the name of the piece (used in auto-reporting moves) will be *Undead Zombie Base* rather than simply *Zombie Base.*

Advanced Layers: When a Zombie unit is in its Undead state, it can activate its Undeath Ray, directed either up, down, or to either Side. We'll add a second Layer to the Zombie Base to represent the Undeath Ray.

- Select Layer again from the list of Available Traits and click **Add**.
- Give the Layer four levels using the images RayN.gif, RayE.gif, RayS.gif, and RayW.gif. Note that these images also use transparency to offset the depiction from the center of the counter.
- The **Increase/Decrease** commands will change the facing of the ray. Set the **Increase** command name to *Rotate Ray CW* and the **Decrease** command name to *Rotate Ray CCW*. (Set the hotkeys for these commands to Ctrl-X/Ctrl-Z so as not to conflict with the commands to rotate the ship.)

Copy/Paste: The Zombie Warship is similar to the Base, except that the ship can change facing. You can save a lot of time defining counter by using the Copy/Paste commands in the Configuration Window.

- Right-click on the Zombie Base component and select **Copy**, and then right-click on the **Zombies [Scrollable List]** component and select **Paste**. Now we need only edit the copy and change a few things.
- Edit the Basic Piece Properties and change the name to *Zombie Warship*.
- Edit the Properties of the first Layer: select **Image 1**, double-click on the image, and select the ZombieWarship.gif file.

Partial Rotation: The order of Traits in a Game Piece is important. Generally, a Trait can modify only those other Traits that appear before (above) it in the list of Current Traits.

- Edit the Zombie Warship and add a Can Rotate Trait.
- Then select it, and click the **Move Up** button until the Trait is between the two Layer Traits. This will make the Zombie Warship depiction rotate without making the Undeath Ray depiction rotate.

Invisibility and Masking: The Invisible Trait enables a player to completely hide a counter from another player. The Mask Trait allows one player to hide details of a counter from another player. The Zombie Minefield will make use of both of these Traits.

- Add another Single Piece to the Zombies Scrollable List.
- Leave the Basic Piece image blank and set the name to *Minefield*.
- Add a Layer with 3 levels, using the mine6.gif, mine8.gif, and mine12.gif images.
- Add a Mask Trait. Set the Mask command to *Reveal* and the keyboard shortcut to Ctrl-R.
- Set the **View When Masked** to the mine.gif image. The Fuzzy player will see only this image until the minefield is revealed. The display option determines how the Zombie player will see the counter. We'll select the *Inset* style, which displays the masked image in the upper left corner as a reminder to the Zombie player that the piece is not revealed.
- Finally, add the Invisible Trait. Under **Can Be Hidden By**, select *Any of the Specified Sides*. Enter *Zombies* and click **Add**. When activated, the counter will be completely invisible to the Fuzzy player. The zombie player will see a transparent version of the piece against a colored

background. Select black for the background color. The Zombie player can make the piece invisible and masked in the Game Piece Palette before dragging it onto the map.

Prototypes

Prototypes are a way of allowing many pieces to share a common set of Traits. In *Zap Wars*, every Zombie unit has the Undeath Ray capability. While Copy/Paste can be used to create the units initially, it can be difficult to manage if the module author later decides to make some alteration that affects many different pieces.

- Right-click on the **[Game Piece Prototype Definition]** node and select **Add Definition.** The dialog for defining a Prototype is the same as the one for defining a Game Piece, but with a name, and without the Basic Piece.
- Define an Undeath Ray layer just as it exists in the Zombie Base and Warship. (You can create this as you did earlier, or you actually open the Zombie Base unit, copy the existing Undeath Ray layer, and then paste it into the dialog for the Prototype.)
- Name the Prototype Definition *Zombie*.
- Edit the Zombie Base and Warship and replace the Undeath Ray layer with a Prototype Trait, using the name *Zombie*.

Now other ship types may be added that use the same prototype. The Undeath Ray layer can be adjusted later, affecting all of the units at once. Furthermore, a new Trait may be added to all pieces at once by simply adding the new Trait to the Prototype definition.

Dice Button

We need to add a Dice Button so we can resolve battles. Right-click the **Zap Wars [Module]** node and pick **Add Dice Button.** We change the **Name** and **Button Text** to *2d10*. In **Number of Sides Per Die**, we enter 10. Because the results of each individual die don't matter, we select **Report Total.**

A button labeled 2d10 is now shown in the Main Controls Toolbar. Clicking it will return the total of a 2d10 roll.

Not all component changes are refreshed in real time. It's a good idea to restart the Module Editor after making major changes to your module, so you can see the changes implemented.

Next Steps

The *Zap Wars* module is well underway now. We can continue to add components to refine the game. Perhaps a Zoom Tool for the strategic display will help view the map better and more clearly. A Line of Sight Thread would be helpful to quickly measure distances on the Tactical display. Experiment until you've created the *Zap Wars* module to your liking.

Card Game

Besides traditional board games, VASSAL can be used to play card-based games, or games that are

mixes of both card and board game. In this tutorial, we will go through the steps for making a VASSAL module for a pure card game called *Raj*.

Raj is a bidding game for up to 4 players. Each player maintains a hand of Cards with values from 1 to 15, and bids for a set of tiles that are revealed one at a time.

Data for this module is in the File:Raj.zip file.

Getting Started

Launch VASSAL. In the Module Manager, pick **File | New Module.** A new, empty module is created, shown here.

We right-click the **[Module]** node, and in the resulting dialog, enter the following:

- **Game Name:** *Raj*
- **Version Number:** *1.0*
- **Description:** *A Bidding Card Game*

We can now begin creating the module.

Sides

To keep Cards clear, we need to specify what Sides are available for players in the game.

- Right-click the **[Definition of Available Sides]** node.
- In the box, type *Red*, and click **Add**. Do the same for *Green, Blue*, and *Purple*.

When players load a saved game or join one on the live server, they'll be prompted which Side they want to take, or whether they just want to be an observer.

Boards

We will make one Map Window for the main playing area: this will be where the tiles are revealed and each player□s bid Cards are placed. In addition, we will make one window for each player to hold his current hand of Cards in.

The Playing Area

Since each module begins with a Map Window by default, we'll make that one into the playing area.

- Double-click on the **[Map Window]** node. For **Map Name**, enter *Playing Area*. You can leave the other settings at their default values for now.

Now we□ll make the playing area blank, but with a definite size.

- Right-click on the **[Map Boards]** node and select **Add Board.**
- In **Board Name**, enter *Playing Area*. Set board width and height to 800x800. In **Background color,** click the color selector and pick a gray or light blue color.

Windows for Player Hands

Now for each Side, we'll create a window for that player's hand of Cards.

- Right-click on the **[Module]** node and select **Add Player Hand.**
- Under **Belongs to Side**, enter *Red*, and then click **Add**. Only the Red player will be able to access the contents of this window.
- In **Map Name**, enter *Red's Hand.*
- Leave the **Visible to Other Players** box unchecked. This will mean that other players won't even see the window.
- Leave the rest of the fields blank. It's possible to give these windows an image for a background by specifying a board, but we'll simply leave the background blank. Click **Ok**.

You can use the Red's Hand window to quickly create the windows for the other players.

- Right-click the **Red's Hand [Player Hand]** node and pick **Copy**.
- Select the **[Module]** node, right-click, and pick **Paste**. Repeat this two more times, for a total of four **[Private Hand]** nodes.
- Double-click one of the copies. Under **Belongs to Side**, Red is listed to the right. Select Red and click **Remove**. Now enter *Green* in the text box and click **Add**. Green will now be able to access this window.
- In **Map Name**, enter *Green's Hand*, and then click **Ok**.
- Repeat these steps for Blue and Purple.

Making the Cards

VASSAL board games draw counters from the Game Piece Palette, with an unlimited supply of each counter. This is not appropriate for Card games. Right-click on the **[Game Piece Palette]** node and pick delete to remove it from your Raj module.

Decks of Cards must be added to a Map Window. Cards added to a Deck in the Configuration Window will be in the Deck when a game is begun. Players click on a Deck to drag the top Card to their hands or a playing area. Right-clicking on a Deck lets players turn it face-up or face-down, shuffle it, or reverse the order of Cards in it.

For this module we will create one Deck that contains the tiles the players are bidding for and one Deck for each player's set of Cards.

- The Deck of tiles goes in the middle of the playing area. Right-click on the **Playing Area [Map Window]** map node and select **Add Deck.**
- For **Name**, enter *Tiles*.
- For **X Position** and **Y Position**, use *400* and *400*, which will put the Deck in the center of the Playing Area.
- Put the **Tile Deck** in the middle of the map, at X Position 400,400.
- The **Width/Height** of the Deck is only used when the Deck is empty, so that players can place

Cards back into the Deck. We'll use the size of one of our tiles, 70x94.

- Click **Ok**.

Now right-click on the **Tiles [Deck]** node and select **Add Card** to add the first Card to the Deck.

Cards in VASSAL are built the same way as counters. The simplest Card is a Basic Piece with the Mask Trait. The image of the Basic Piece will be the front of the Card and the image for the Mask will be the reverse of the Card.

- Two Traits are listed under Current Traits: Basic Piece and Mask.
- Double-click *Basic Piece*. For the **Name**, enter Card 1. Double-click on the left Side of the dialog, and browse to the tile1.gif image in the rajData directory. Click **Ok**. You have now defined the name and the front image for the Card.
- Double-click *Mask*. In **Display Style**, pick *Background*. For **View When Masked**, double-click the white area and browse to tileBack.gif in the rajData directory. Click **Ok**. This defines the Card back.
- If we needed more features for our Cards, such as the ability to rotate them Sideways or place markers on them, that could be done by adding more Traits. They would go above the Mask Trait if you wanted them to only show when the Card is face up. However, we only need simple Cards for this tutorial.
- Now right-click on the Tile you just made and pick **Copy**. Then, on the **[Deck]** node, click **Paste**. This will make a copy of the first Card.
- Double-click the copy. In Basic Piece, change the name to Card 2, and pick tile2.gif for the Card front.
- Repeat for each of the other 13 tiles (15 in all).

Now you are ready to create each player□s Deck. For the tiles we needed a different image for every Card. We can save some steps when creating the players' Decks. The players' Cards are simply numbered 1-15 on the front, so we'll use a Text Label Trait to write the number on a common background image.

Right-click on the **Playing Area [Map Window]** node to create another Deck.

- We'll name this Deck *Red□s Cards* and put it at 400,150 with size 150x240.
- Right-click to add a new Card. Use FrontRed.gif for the front and RedBack.gif for the back.
- Now select *Text Label* from the list of **Available Traits** and click **Add**. This will be a permanent label, not changeable during the game, so set the **Text** to *1* and make the **Menu Command** blank.
- Set the **Font Size** to 52. Set the **Text Color** to black and the **Background Color** to white.
- Set the vertical and horizontal position and the vertical and horizontal justifications all to *Center*. Click **Ok**.
- Now with the Text Label selected in the list of Current Traits on the right, click **Move Up** until the Text Label Trait is above the Mask Trait. This will ensure that the number is not showing when the Card is face down.

- Now right-click on the Card you just made and pick **Copy**. Then, on the **Red□s Cards [Deck]** node, click **Paste**. This will make a copy of the first Card.
- Double-click the copy. Edit the Text Label Trait and change the text of the label from 1 to 2.
- Repeat the **Copy**/**Paste**/**Edit** process for each of the other 13 tiles (15 in all).

Having made the Red Deck, the others follow quickly.

- Right-click on the **Red□s Cards [Deck]** node and pick **Copy**. Select the **[Module]** node and pick **Paste**. This copy will become the Green Deck.
- Double-click on the copied Deck. Set its Name to **Green□s Cards**, and set its position to 700,400.
- VASSAL provides a convenient feature to edit many pieces at once. Right-click on the Green Deck and select **Edit All Contained Pieces**. You'll see the Properties window for the first Card, but all changes you make to Traits in this window will apply to all Cards in the Deck. For the Mask Trait, set the front image to FrontGreen.gif and the back to GreenBack.gif.
- Repeat the **Copy**/**Paste**/**Edit** process for the Blue and Purple Decks.

During Play

To play, one of the players turns the first tile in the Deck face up. Then each player selects a Card from his hand, turns it face down, and drags it to the playing area. All players then reveal their Cards simultaneously. The highest Card wins the tile, but Cards of the same value cancel each other out.

For example, the players play 12, 6, 8, and 12. The 12s cancel, so the 8 wins. The playing buying the tile moves it to his area in the playing area and the used Cards are deleted. After all tiles have been bought, the player with the highest tile total wins.