GETTING STARTED WITH VASL DEVELOPMENT

A STEP-BY-STEP GUIDE

PREPARED BY DOUG RIMMER

DOUGRIMMER@ROGERS.COM

GETTING STARTED WITH VASL DEVELOPMENT

DOCUMENT OVERVIEW

This document provides instructions on obtaining, installing, and using a number of tools necessary to enable VASL development.

This is a multi-step process involving a number of separate tools and products and can appear quite complex. This document tries to provide sufficient detail to allow those interested to get up and running on their own. In the event of problems requiring assistance, contact information is provided.

The document is written to enable those unfamiliar with such tools to create an environment in which they can contribute to further development of VASL. Experienced users will find that they can skip some of the introductory text in each section and proceed directly to the detailed instructions.

Obtaining and installing the tools required for VASL development involves using websites and software that have their own development and update processes. This will often create minor changes in the look-and-feel of these tools. Users should anticipate this and not be stymied if each detailed step is not exactly as described in this document.

For those interested in creating VASL Boards, see separate instructions on the VASL wiki within GitHub (see Section 3 below for information on GitHub). Both GitHub and SourceTree (see Section 3) are recommended for those working on VASL Boards.

GETTING STARTED OVERVIEW

In order to work on VASL source code, a number of steps must be completed. The VASL source code is a library, not a complete standalone application. The library is used by VASSAL (http://www.vassalengine.org), which is a standalone application. Development will require the VASL source code along with VASSAL.

The source code is written in Java. To enable access to this, it is necessary to install a version of the Java SE Development Kit (SDK), which includes tools for developing, debugging, and monitoring Java applications.

To obtain the source code for VASL and to manage changes made to the code, it is necessary to use the GitHub Repository and SourceTree. This requires creating a Git user account, installing software, and using SourceTree to obtain and manage code changes.

To actually develop new code requires an Integrated Development Environment (IDE). Several different IDE's can be used, and this document provides instructions for one of them: IntelliJ. Other IDE's will have help tools that can be used to walk users through the necessary steps.

In order to integrate code changes, it is necessary to 'build' the VASL module. This document provides instructions for doing so using a tool named Maven which has to be obtained and installed.

While many of the above tools can be purchased, all have "community" or free-access versions that can be used for VASL development. No purchases are necessary.

Finally, this document provides more detailed instructions on how to obtain, install and use all of the above-mentioned tools.

1.0 VASL AND VASSAL

1.1 THE VASL MODULE

The VASL source code is a library, not a complete standalone application. The VASL source code is compiled into the VASL module, which is used by VASSAL to provide the ASL game interface via the internet. Usage of the source code is subject to the terms of the Library Gnu Public License (LGPL), described in the LICENSE.txt file and also available from http://www.opensource.org.

1.2 THE VASSAL ENGINE

VASSAL is a standalone application which is used in conjunction with modules such as VASL to provide game interfaces. Neither VASSAL nor any of its modules is a complete game in itself. Both VASSAL and a game module are required in order to play.

1.2.1 VASSAL under the covers: why you need it and what it does

VASL is one of many game modules developed for the VASSAL game engine. The VASSAL module editor is used to add and update VASSAL capabilities, examples of which would include counters, map window options (e.g. zoom), etc.

VASL is one of a small number of modules that contain a lot of custom Java code to enhance and augment standard VASSAL features. For example, concealing a stack is performed via custom VASL code.

VASL "development" could involve either of the above.

1.2.2 VASSAL Module documentation

If you wish to know more about VASSAL functionality and how it works, please consult the VASSAL documentation. There are also a series of YouTube videos about VASSAL, some quite detailed. A solid understanding of VASSAL module development should be considered a prerequisite to VASL development.

1.2.3 Obtaining VASSAL and using it from within the VASL source code

The VASSAL files necessary for VASL code development are included with the VASL source code library and will be copied to your computer with the rest of the source code library (see Section 3.1.4). In order to access the VASSAL functionality from within VASL, several references to VASSAL will be created in Section 4.1.3 Step 2.

1.2.4 Using the VASSAL Module Editor

The VASSAL module editor comes with VASSAL program that you install to play VASL. See the VASSAL documentation for more details. The VASSAL module editor is used primarily for counter development and other non-code related changes to VASL. It is not used for adding custom java code.

2.0 GETTING THE UNDERLYING TOOLS: SDK

2.1 SETTING UP THE JAVA SOFTWARE DEVELOPMENT KIT (SDK)

Section 1.2.1 describes how Java code is used in the VASL module to enhance standard VASSAL features. In order to be able to use Java code from within VASL, a Java Software Development Kit (SDK) must be installed and then referenced. Details on how to reference the SDK in VASL projects are included in Section 4.1.3.

2.1.1 The SDK: why you need it and what it does

The SDK works primarily behind the scenes in VASL development to enable the use of Java code in the VASL module. The SDK includes tools for developing, debugging, and monitoring Java applications which are exposed to the developer via their Integrated Development Environment (see Section 4).

2.1.2 Getting the SDK

The SDK can be found at http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html.

If this link does not work, search for the Java Software Development Kit.

2.1.3 Installing the SDK

Java offers several versions of the SDK. Java SE Development Kit 8u152 is the latest version to be successfully used with VASL. When newer versions appear, feel free to try them. If you encounter problems, install and reference the version cited here.

Click to accept the license agreement and the click on the link which corresponds to your system.

Complete the download and installation of the SDK by following the screen prompts as you would for any software install. It may be helpful to note the directory in which the SDK is installed as you will need to be able to find it when adding Java references in Section 4.

In order to validate the SDK install and to ensure that Java is in the path (this will be needed in Section 6), go to the command line and type "java -version" without the quotation marks. If you don't know how to find the command line, please Google it as there are many ways to get there depending on computer types and OS versions. If typing "java –version" does not return a reference to Java, then you will need to add it to your path manually. Since methods of doing so can vary by OS, please check the internet for details.

Finally, verify that the Environment Variable JAVA_HOME exists on your computer and points to the installation directory. Since methods of doing so can vary by OS, please check the internet for details.

3.0 GETTING THE CODE: GITHUB AND SOURCETREE

The VASL source code is stored in a code repository called Git.

3.1 THE GIT CODE REPOSITORY

A Code Repository is essentially a web-based file storage location that includes tools for managing access to files and updates to them, and is specially designed to support software development by teams of people.

3.1.1 Code Repository: why you need it and what it does

Git is used to store all of the VASL source code. In addition, Git is a version control system (VCS) that records changes to a file or set of files over time so that specific versions can be recalled later. A VCS allows developers to revert files back to a previous state, revert the entire project back to a previous state, compare changes over time, see who last modified something that might be causing a problem, who introduced an issue and when, and more. Using a VCS also generally means that if someone screws things up or loses files, it is possible to easily recover.

A short primer on Git is provided at Annex A.

To access the Git repository, it is recommended that you use the GitHub website and install SourceTree. Both of these tools work with Git to provide access to/manage the VASL repository in Git.

### 3.1.2 How to install Git and SourceTree and access GitHub

3.1.2.1 GitHub

Create a Github account at https://github.com/ by typing a username, email address and password, and then clicking "Sign up for GitHub".

3.1.2.2 Git

Git must be installed on your computer. It can be found at http://www.git-scm.com/ or by googling 'git-scm' if the link does not work. Git-scm is available for most major operating systems. Choose the latest version for your system.

You may see references to the GitHub desktop or GitHub GUI. For the purposes of VASL development, it is NOT recommended that you install this software. SourceTree should be used instead.

When installing git-scm on Windows use the default options with the exception of allowing git commands from the command prompt.
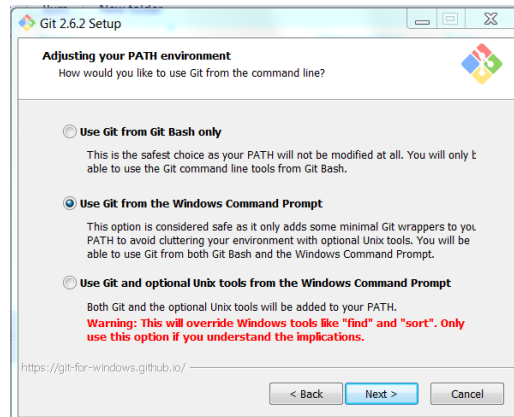
3.1.2.3 SourceTree

1. NOTE: There are many git client tools available, but we are recommending SourceTree as it's one of the easiest to use.
2. Go to https://www.sourcetreeapp.com/.
3. Click on Download SourceTree Free (for Windows. Click on the appropriate links for other OS).
4. Click Run in the dialog box.
5. Follow standard install instructions on screen using the SourceTree Setup Wizard.
6. Launch SourceTree to validate installation. If prompted to create an SSH key, click 'No' as this is not required.
7. While SourceTree is free, you will need to register with them to use it beyond 30 days. Click Menu File → Registration.


3.1.3 Configuring the Code Repository

Following installation (see Section 3.1.2.2), some minor configuration of Git is required.

1. Open SourceTree.
2. Select Menu Tools → Options.
3. Ensure that the "Allow SourceTree to modify your global Git and Mercurial config files" is checked.
4. Enter a username in the Full Name textbox. Use the same name as the username provided when creating your GitHub account (see Section 3.1.2.1). The email provided should be the same one used to create your GitHub account (see Section 3.1.2.1).
5. Enter an email address in the Email Address textbox.

3.1.4 Cloning the VASL code repository for local development

1. In order to update the VASL code repository, it is necessary to join the vasl-developers team on GitHub. To do so, go to https://github.com/orgs/vasl-developers/people and message one of the owners that you would like to join. (This is only necessary if you wish to have "write" access to the repository so that you can contribute code. If you just want to clone a repository and code and build separately, you don't need to join the team.)

2. Once, you have received confirmation that you are a member of the group, sign in to GitHub.

3. Search for "VASL".

4. Select "vasl-developers/vasl" (click on the 'vasl' at the end as it is a separate hyperlink).

5. Click on Branch-develop and type a name for a new branch in the textbox then click on "Create Branch:". If "Create Branch:" does not appear, check that you are a member of the vasl-developers group. For those wishing to develop and contribute code, please start your branch name with the prefix "feature/".

6. Copy the HTTPS Clone URL for your branch to the Clipboard by clicking on the "Cone or download" button on GitHub and copying the URL for the text box that will display.
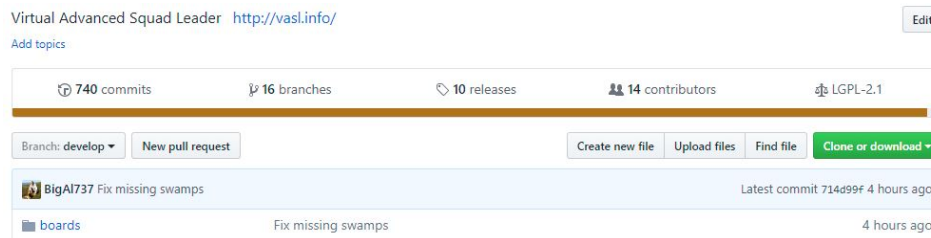
7. Start SourceTree. (As an alternate to steps 7-12, it is possible to clone the VASL code repository from within the IDE (see Section 4.1.4). However, it is strongly recommended to use SourceTree.)

8. Click the Clone/New button (or, depending on the version of SourceTree, click the File menu and then select Clone/New).
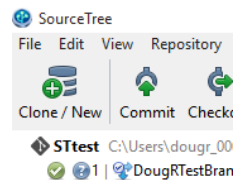
9. Paste the HTTPS Clone URL for your branch into the "Source Path / URL" textbox.

10. Select a destination location by typing in the text box or clicking on the Browse button (which will allow you to create a new folder if desired). It is recommended that you use a directory where your IDE will store projects.

11. Click on the Advanced Options dropdown arrow. Enter the name of your new branch into the "Checkout Branch" textbox.

Clone

Cloning is even easier if you set up a remote account

| Source Path / URL: | | Browse |

Repository Type: ❓ No path / URL supplied

| Destination Path: | | Browse |

| Name: | |

Local Folder:

| [Root] | ⌄ |

⌄ Advanced Options

Clone

12. Click Clone.

The cloning process will take several minutes depending on your computer.

4.0 WORKING WITH THE CODE: INTELLIJ

4.1 THE INTELLIJ IDE

Integrated Development Environments (IDE's) provide a means of linking a number of tools together to allow developers to write software code and then manage their work by linking to project management tools and repositories.

4.1.1 IDE's: why you need one and what they do

Above all, the IDE is the place where code gets written. IDE's link to underlying software development tools (such as the SDK) that then provide developers with access to coding languages, structures and libraries. They also integrate tools that allow developers to link their work to larger teams and projects.

A number of IDE's can be used to write VASL source code in Java, for example, Eclipse. The following instructions apply specifically to another one, IntelliJ. They could be used as a guide to working with Eclipse or other IDE's.

### 4.1.2 Installing an IDE: IntelliJ

To install IntelliJ:

1. Go to https://www.jetbrains.com/idea/download/

2. Download the Community Edition version for appropriate platform (Windows, Mac, etc.)

3. Run install as normal for the platform (i.e. on Windows, click "Open" or "Run" in the dialog boxes)

### 4.1.3 Setting up IntelliJ for development

1. Create the VASL project

> The following instructions assume that you cloned the VASL code repository using SourceTree as per Steps 7-12 of Section 3.1.4. If you intend to clone VASL from within IntelliJ, proceed to Section 4.1.4.
>
> Start IntelliJ. You should see an introduction screen with the option to import a project. Select that option. The screen may have differences from the one shown below due to version changes but should still have an Import Project option. Click it.
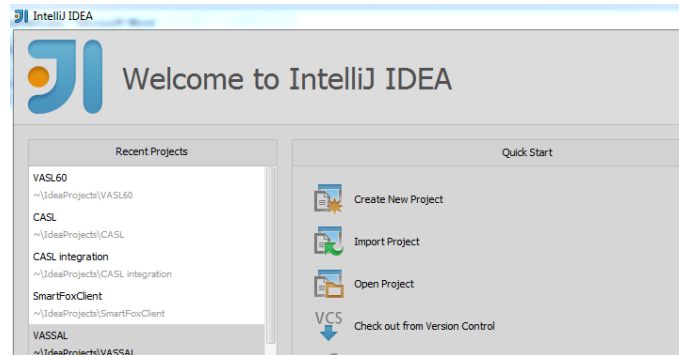
> Navigate to the folder where you stored the cloned branch from Git (see Section 3.1.4 Step 8) and click OK.
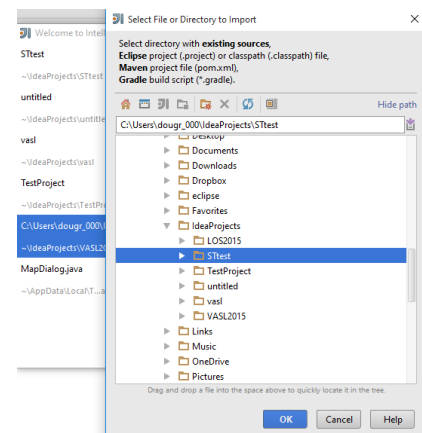

> IntelliJ will then present a number of screens to complete the importation process. These screens change often and thus are not shown or described in detail. For the most part it is ok to accept the default choices present by IntelliJ. A few key points to check are highlighted below.
>
> On a screen labelled something like "Import Project", ensure the "Create project from existing sources" button is selected then click Next. If shown a project name and location, accept the defaults.

If shown a screen with a Libraries list, ensure "lib" is selected. Deselect "vassal-3.2.8-sources" if it is selected. If shown a screen with a "Modules" list, select those shown. If shown a screen about the Java SDK, select the most recent SDK available (and that you have installed; likely 1.7 or 1.8). Note that if you installed Java SDK 8.0 in Section 2.1.3, this will show up as 1.8.

At the end of this process, IntelliJ may give you the following messages:
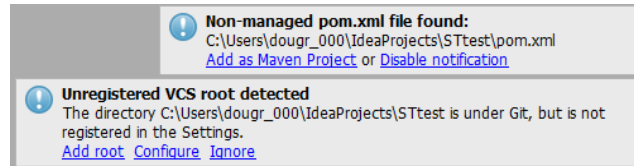
Click on Add as Maven Project and Add Root. If one or both messages disappear before you can click on them, they can be viewed in the event log (Menu View → Tool Windows →Event Log), where they can be clicked.

2. Configure the VASL project

You will need to configure the project in order to enable tools such as the SDK and VASSAL code to be accessed when coding.

Select Menu File →Project Structure. Select Project Settings and check that Project SDK is the one that you installed (see Section 2.1.3). If not, click the dropdown or "New" button to select the Java SDK.

Click on the Project language level dropdown and select "5.0".
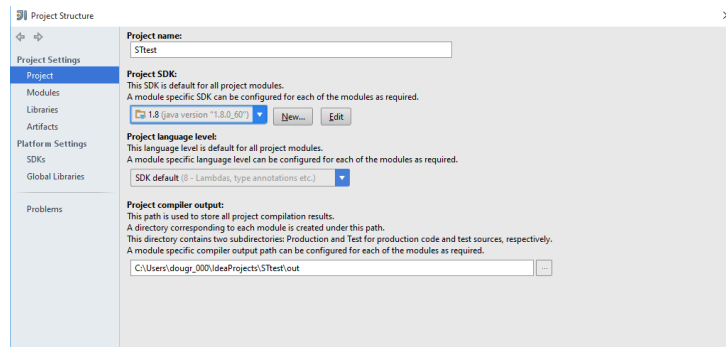
Click the Modules tab and then the Dependencies tab. The list should include the following items.
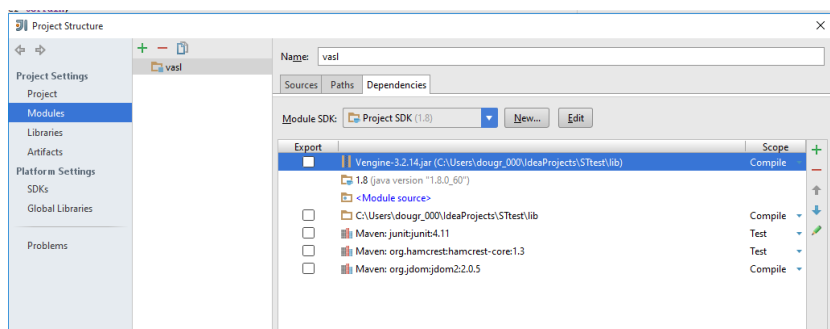
If the Java item is missing, click the green + sign and select "Library"→ "Java" and navigate to the item, then click OK. If you have trouble finding the SDK, look in directories where software files are usually kept (e.g. directories with "Program Files" in their name in Windows).

If the Vengine item is missing, click the green + sign and select "Jars or Directories". Then navigate to the "lib" directory of the current project and select "Vengine.jar (version numbers might be different), then click OK. Select "Vengine.jar" and use the up and down arrows to move it to the very top of the list.

If the lib item is missing, click the green + sign and select "Jars or Directories". Then navigate to the "lib" directory of the current project and click OK.

These previous two steps must be done separately and the "Vengine.jar" item MUST be at the top of the list.

To ensure that the link to Git is in place, select Menu Settings and on the next screen select Version Control and then Git. Verify that the correct path to your installation of Git (see Section 3.1.2.2) is listed in the Path to Git executable textbox. If not add it by typing the path or navigating to it via the Directory box beside the textbox. Click Test to verifying the linkage.
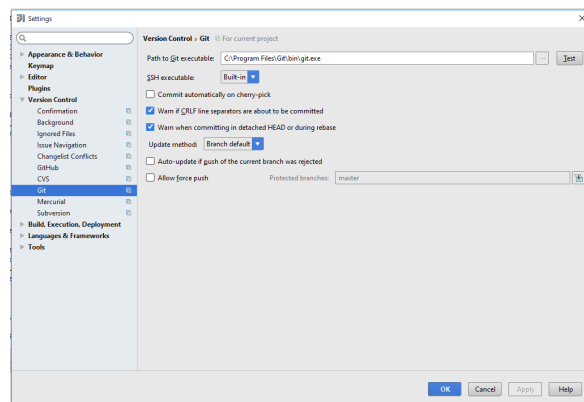
For GitHub, click on the GitHub tab on the same Settings screen and enter your username and password for GitHub. Make sure Host shows GitHub.com and click Test to verify the connection is enabled and then click OK.
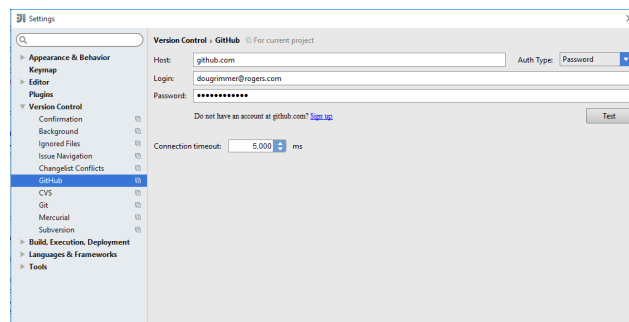

To run the project, (to verify that code changes are working), certain Run configurations must be set. Select Menu Run → Edit Configurations. If no item is listed in left-hand listbox,

click the green + sign and select "Application". Enter a name in the Name textbox and "VASSAL.launch.Player" in the Main Class textbox.

Enter --standalone "C:\Users\<you>\IdeaProjects\<your project name>\target\vasl-6.4.1.jar" in the Program Arguments textbox, using the directory where your current module file is located. Note that versions numbers ("6.4.1") will change over time. This must point to an actual .jar file so when running a module for the first time it is necessary to either build the module first (see Section 6.1) or point to already existing file (such as the one used to play VASL games).

Ensure that the Working Directory textbox refers to the folder where the current project is located. If not, type in the folder name or navigate to it using the Directory button beside the textbox. Select the classpath of module from choices available from the dropdown arrow; do not select "test". Instead use the option similar to your project name.
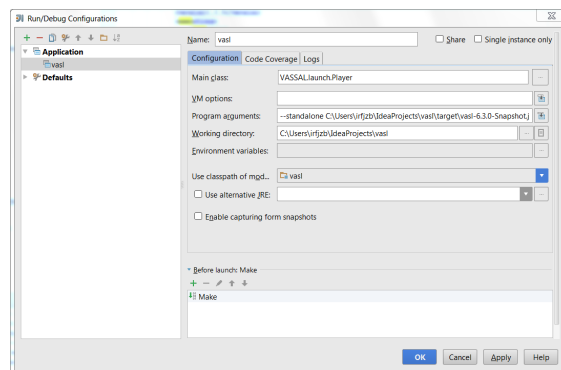

Click OK.

Select Menu Run → Run "Your project name" to run the project.

4.1.4 Cloning the VASL code repository for local development from within IntelliJ

1. In order to clone the VASL code repository from within IntelliJ, follow the steps in Section 4.1.3 but select the "Create New Project" option in Step 1 rather than Import Project. Some of the screens in the remainder of 4.1.3 Step 1 may contain minor differences.

2. Click Menu VCS → Enable Version Control Integration. You must first enable integration of Git in IntelliJ before you can check in changes. Once enabled, IntelliJ, via Git, will track all changes in your project directory and make them available to check in.
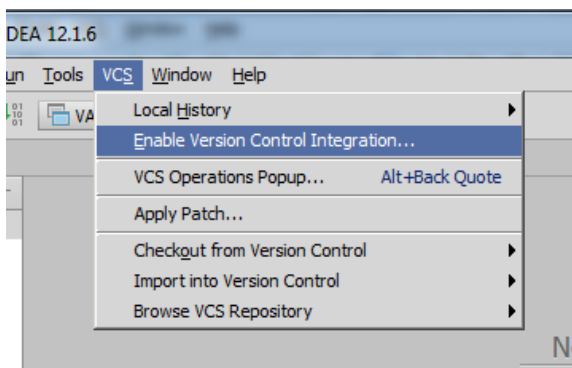
In the dialog select Git from the Dropdown:
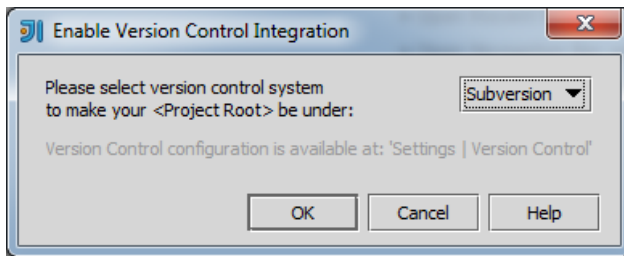
3. Click Menu VCS → Checkout from Version Control → GitHub and follow the required steps.

5.0 MANAGING THE CODE: MORE SOURCETREE

5.1 USING SOURCETREE AND THE IDE TO CONTROL CODE CHANGES

While writing new code is one thing, ensuring that it integrates properly with the existing code base and new code being developed by other developers on the team is another issue. The IDE and SourceTree work together to do so.

5.1.1 Code Management: why we need it and what you must do

Good code management ensures that no work is wasted, and no conflicts enter the code base due to uncoordinated efforts of multiple developers. All developers must access the VASL code and commit changes properly in order to maintain an integrated set of code files that will produce the VASL module for use by ASL players.

5.1.1 Creating a branch

Developers should create a branch and use it for specific coding tasks before proceeding to write code. Branches should be created before cloning the source code from Git to SourceTree (or directly into IntelliJ). See Section 3.1.4 Step 5 for instructions.

5.1.3 Committing changes to your branch

Once you have cloned the code repository and created project within the IDE, you can now make changes to the code.

To integrate those changes into your branch in the VASL code repository in Git:

5.1.3.1 If you used SourceTree to clone the repository:

1. Update your desktop project to be even with develop. Looking at your branch on GitHub. If there is a box indicating that your branch "is x commits behind develop" then you MUST update your desktop project first before adding your changes to GitHub – see Section 5.1.3.2 Step 5.

2. Open SourceTree and ensure that the cloned code branch you created in Section 3.1.4 Steps 7-12 is selected.
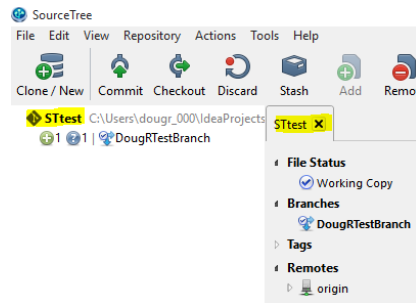
3. In the Project Window, click on the File Status Tab.
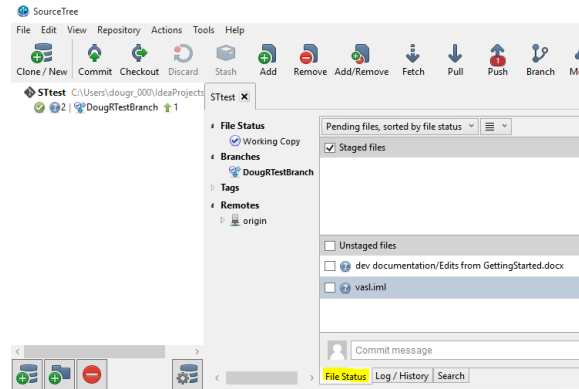
4. Files ready to commit should be listed under Staged Files. If the files you want to commit appear under Unstaged files, click the checkbox beside the file to move it to Staged Files. Clicking the checkbox beside Unstaged files will move all the unstaged files to Staged Files at once.

If for some reason the "pom.xml" file shows in the list of Staged or Unstaged files, leave it as an Unstaged file. Other files with ".iml" suffixes can also be ignored. There is no need to push these files up to GitHub. Any files that begin with ".idea" or "target" can also be ignored and should not be pushed to GitHub.

5. Before committing, please add a commit message in the textbox below the Unstaged files list. You can see examples of commit messages in Git repository and should put the issue number (E.g. #12) in the commit message. If you do Github will automatically link the issue to the commit. This is very handy. Here's an example: https://github.com/vasl-developers/vasl/pull/227.

6. Click "Commit".

7. The committed files are now ready to be "Pushed" to your branch in the Git repository. Click Push.

It is not necessary to immediately "Push" committed files. They will stay committed in SourceTree until pushed or uncommitted.

The Fetch, Pull and Merge buttons allow you to update your branch with changes that others have made to the VASL source code in separate branches. Alternately, you can bring changes to the VASL source code on GitHub directly into your IntelliJ project – see Section 5.1.3.2 Step 5.

5.1.3.2 If you cloned the VASL source code from within IntelliJ (or another IDE) as per Section 4.1.4:

1. Click Menu VCS → Commit Changes.

If you wish to add new files to the repository, first right-click on the file in the project browser and use the "Git →Add" menu option, then Commit Changes.
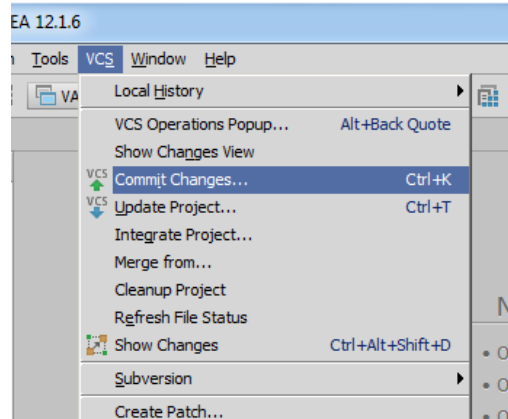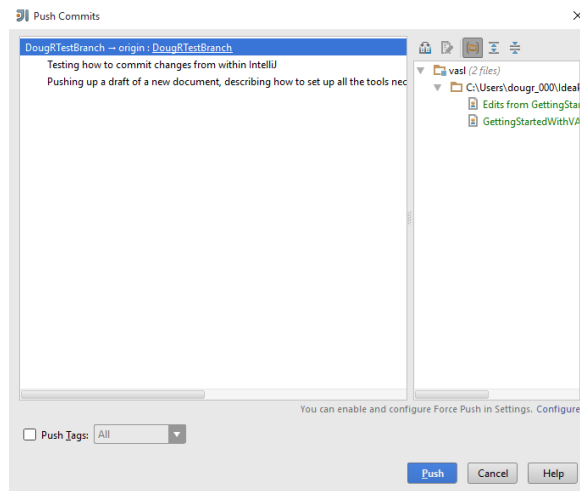

2. Ensure that the files you have changed and wish to commit show up in the Commit Changes list. Be sure to add a Comment to enable others to understand what changes you are committing. Developers should include the issue number in the commit message.
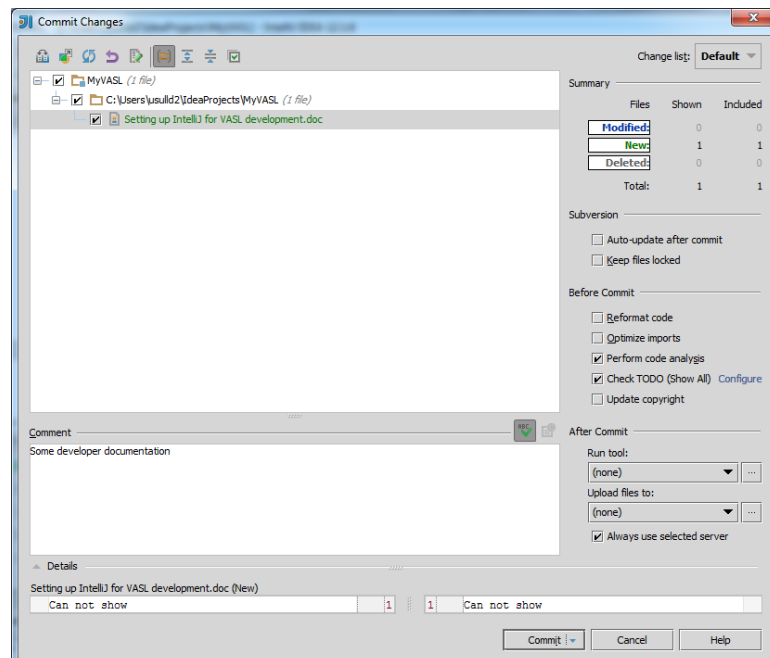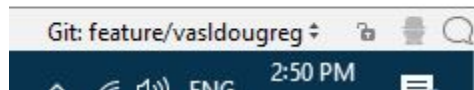

3. Click Commit.

4. Select Menu VCS → Git → Push to push the changes to your branch on the Git Repository.

Alternately, you can use the Dropdown on the Commit button in Step 3 to select Commit and Push and do both steps at the same time.

5. To obtain updates from GitHub, select the Git drop down, usually found in the bottom right corner of IntelliJ.

This will bring up a list of GitHub branches. Select origin/develop and then click Merge into Current from the popup menu.

6.0 BUILDING THE VASL MODULE: FROM MAVEN

6.1 BUILDING THE VASL MODULE

Building the VASL module is not required to test the impact of your changes on the overall code. You need to build the VASL module if you modified any of the files in the "dist" folder (note there's no code there). All of the images are in there. If you add counters, new images will be added, and the build file will be modified to add the new counters to the counter tray.

6.1.1 Building VASL: why you need it and what you need to do

You should build the VASL module if you added or changed files in the "dist" folder. This requires the use of software called Maven, which must be installed and then used to create the VASL module. You can also build the VASL module to provide a source module for the Configurations settings in your IDE – see Section 4.1.3 Step 2.

6.1.2 Getting and setting up Maven

Maven can be downloaded from https://maven.apache.org/download.cgi. Once downloaded, click on the Install link on the page and then follow the install instructions. While not difficult,

there are several particular steps that must be followed, and it is preferable to use the instructions on the Maven site.

For convenience add the Maven bin folder to your Path. This will allow you to build the VASL module from the command line and the terminal window in IntelliJ. To confirm Maven has been properly installed type "mvn –v" at the command line to display the installed version. Adding items to your computer's Path varies depending on OS so consult the Internet for details.

6.1.3 Building the VASL module from Maven

1. After installing Maven, copy the "settings.xml.TEMPLATE" document found in the directory to which you cloned your VASL branch in Section 3.1.4 Step 10 (or Section 4.1.4) to the .m2 directory in your home folder and rename to "settings.xml". Your "home folder" may vary depending on OS. For Windows users, for example, look in the "C:\Users\YourUserName" directory.

2. Rename the file in the .m2 directory to "Settings.xml". Open the file, using a text editor such as Notepad on Windows. Edit the file by changing the path to your local JDK installation (see Section 2.1.3).

3. Go to the command line. How you do this will depend on your operating system. Check its help system or the web for details. In Windows, for example, press the Winkey + R then type cmd in the dialog box that appears.

4. At the command line, change the directory to your project directory. To do so in Windows, type "chdir DirectoryName" at the command prompt where DirectoryName is the project directory (full path required).

    Example: chdir C:\Users\dougr_000\IdeaProjects\STtest

5. To create the VASL module, type "mvn clean install" at the command prompt. This process may take several minutes. If successful, the final messages displayed will be similar to this:

    ```
    [INFO] ------------------------------------------------------------
    [INFO] BUILD SUCCESS
    [INFO] ------------------------------------------------------------
    [INFO] Total time: 24.289 s
    [INFO] Finished at: 2015-10-23T12:58:02-05:00
    [INFO] Final Memory: 15M/351M
    [INFO] ------------------------------------------------------------
    ```

6. The build creates a .jar file in the \target directory of the project directory referenced in step 4. This .jar file can be used to in the "Program arguments" text box referred to in Section 4.1.3.

7. In order to run the VASL module, type "mvn exec:exec" at the command prompt (from the VASL directory – see Step 5).

Annex A

A Git primer

Below is a short primer on Git, based on "Getting Started with Git", a more detailed resource found on the Git-scm website at http://git-scm.com/book/en/v2/Getting-Started-About-Version-Control.

As mentioned above, Git is a version control system that allows team development of software code while managing code updates, code conflicts, and rollbacks of changes if required. As a distributed VCS, Git provides each user with a set of code files that fully mirror those in the specific Git repository being used.

As a result, most operations in Git only need local files and resources to operate. Because you have the entire history of the project right there on your local disk, most operations seem almost instantaneous.

Once the VASL source code repository is downloaded to a local computer (see Section 3.1 Step 5), Git has three main states that the local files can reside in: committed, modified, and staged. Committed means that the data is safely stored in the local database. Modified means that the file has been changed but not yet committed to the local database. Staged means that a modified file in its current version has been marked to go into the next commit snapshot.

This leads to the three main sections of a Git project: the Git directory, the working directory, and the staging area.

The Git directory is where Git stores the metadata and object database for a project. This is the most important part of Git, and it is what is copied when a repository is cloned from another computer.

The working directory is a single checkout of one version of the project. These files are pulled out of the compressed database in the Git directory and placed on disk for use or modification.

The staging area is a file, generally contained in the Git directory, that stores information about what will go into the next commit. It's sometimes referred to as the "index", but it's also common to refer to it as the staging area.

The basic Git workflow goes something like this:

1. Modify files in the working directory.
2. Stage the files, adding snapshots of them to the staging area.
3. Do a commit, which takes the files as they are in the staging area and stores that snapshot permanently to the Git directory.

If a particular version of a file is in the Git directory, it's considered committed. If it has been modified and was added to the staging area, it is staged. And if it was changed since it was checked out but has not been staged, it is modified.

There are a lot of different ways to use Git. There are the original command line tools, and there are many graphical user interfaces of varying capabilities. Some knowledge of Terminal in Mac or Command Prompt or Powershell in Windows would be useful.

Update History

September, 2016	Minor text edits
March, 2017	Minor text edits related to Idea IDE changes
March, 2018	Minor text edits related to Idea IDE changes
June, 2019	Clarifications and updates
April, 2020	Clarifications and updates