Phil Shinn

IVR Design Group  110 River Dr., 1103, Jersey City, NJ

VUID ToolKit

Version 2.0

August 2015

Contents

[Introduction 2](#_Toc428526998)

[System Overview 2](#_Toc428526999)

[Requirements 3](#_Toc428527000)

[Getting Started 4](#_Toc428527001)

[Custom Shapes 9](#_Toc428527002)

[Start 9](#_Toc428527003)

[Prompt State 9](#_Toc428527004)

[Arc (Arrow) 10](#_Toc428527005)

[Grammar State 10](#_Toc428527006)

[Computational State 11](#_Toc428527007)

[Decision State 11](#_Toc428527008)

[Stop/Return 11](#_Toc428527009)

[Sub-Dialog State 12](#_Toc428527010)

[Grammar 12](#_Toc428527011)

[Example Design 13](#_Toc428527012)

[Editing/Adding Custom Properties 15](#_Toc428527013)

[Sharing a Design 17](#_Toc428527014)

[Export to HTML 17](#_Toc428527015)

[Export to/Import from XML 17](#_Toc428527016)

[Cloud Sharing with Draw.io 17](#_Toc428527017)

# Introduction

This document describes the Voice User Interface Designer Toolkit (VUID Toolkit or VTK), a collection of tools that supports designing and maintaining a voice user interface (VUI). The original code was written by Phil Shinn and Matt Shomphe, and placed into the public domain in 2007. Our goal was to make life easier by helping to set up a uniform standard markup language for VUI designs. Since then, a number of organizations have adopted the toolkit for their own use.

# License

The code is licensed under the GNU General Public License version 3 (GNU GPLv3), which is described here: <http://www.gnu.org/licenses/quick-guide-gplv3.html>. These are the terms: “This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version. This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details”

Yes, the code is free, as in free beer. If you would like support, contact Phil at [phil.shinn@gmail.com](mailto:phil.shinn@gmail.com) or call +1 (646) 736 -2377.

# Overview

You can use VTK to design an interactive application, just as you could with Visio. You could stop there, or, use some of the simpler ‘helper’ applications in the toolkit to do things like extract the prompt list or the grammar list.

In addition, you can use some of the other tools to develop the suite of test cases associated with the application, so if you change the GUI, it is relatively easy to update the test cases.

Finally, this version of the toolkit also supports an interactive mode, which allows you to run the application as a text-based application, based on the design you created.

# Requirements

The current version has been tested on Windows 7, but should work on XP and all later versions of Windows.

Chrome, the Google browser, which can be downloaded here: <https://www.google.com/intl/en/chrome/browser/desktop/>

Draw.io Desktop is used for the GUI. To get the desktop version, go to the Chrome web store and download the Draw.io Desktop (<https://chrome.google.com/webstore/detail/drawio-desktop>). This is a free plug-in to Chrome that replaces Visio[[1]](#footnote-1).

Python 2.7 available for free download here: <https://www.python.org/downloads/>. Do not use Python 3.0 (yet).

The Python Windows extensions available for download here: <http://sourceforge.net/projects/pywin32/> <TBD make the VTB Python code into an exe>

The VTK legend, document template and starter app which can be downloaded here: <TBD, named VTK 2.12.xml>

The VTK code for executing the design, which can be downloaded here: <TBD, named VTK\_GUI\_2.001.py

# Getting Started

After you’ve got Chrome, and have installed the Draw.io plug-in, you can launch Chrome/Draw.io via the Google Chrome App Launcher:

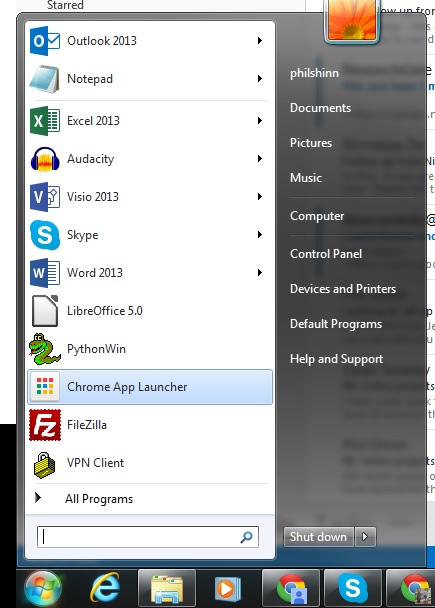


Figure 1 Chrome App Launcher

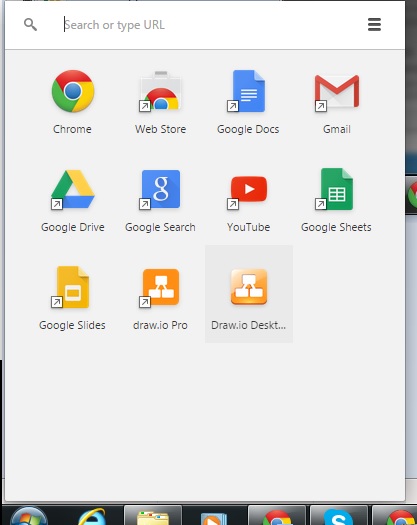


Figure Selecting Draw.io Plugin

Select Draw.io Desktop and launch it:

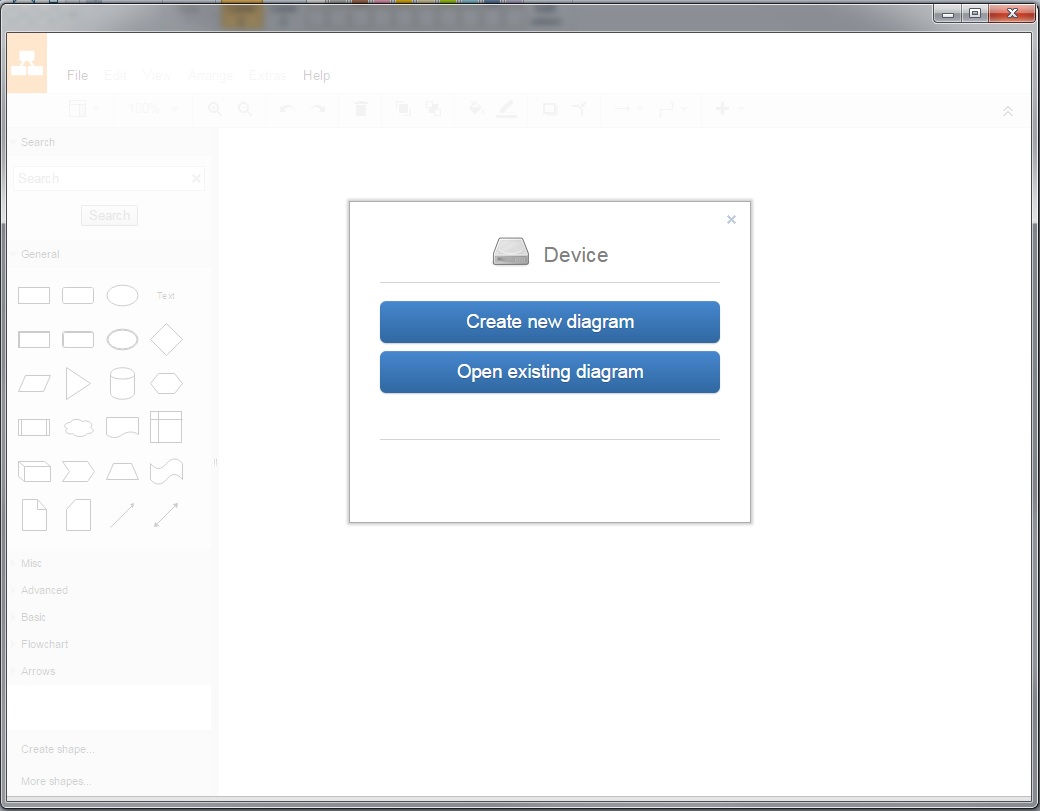


Figure 3 Draw.io desktop Launched

Drag the file VTK 2.0.xml into the Draw.io canvas:

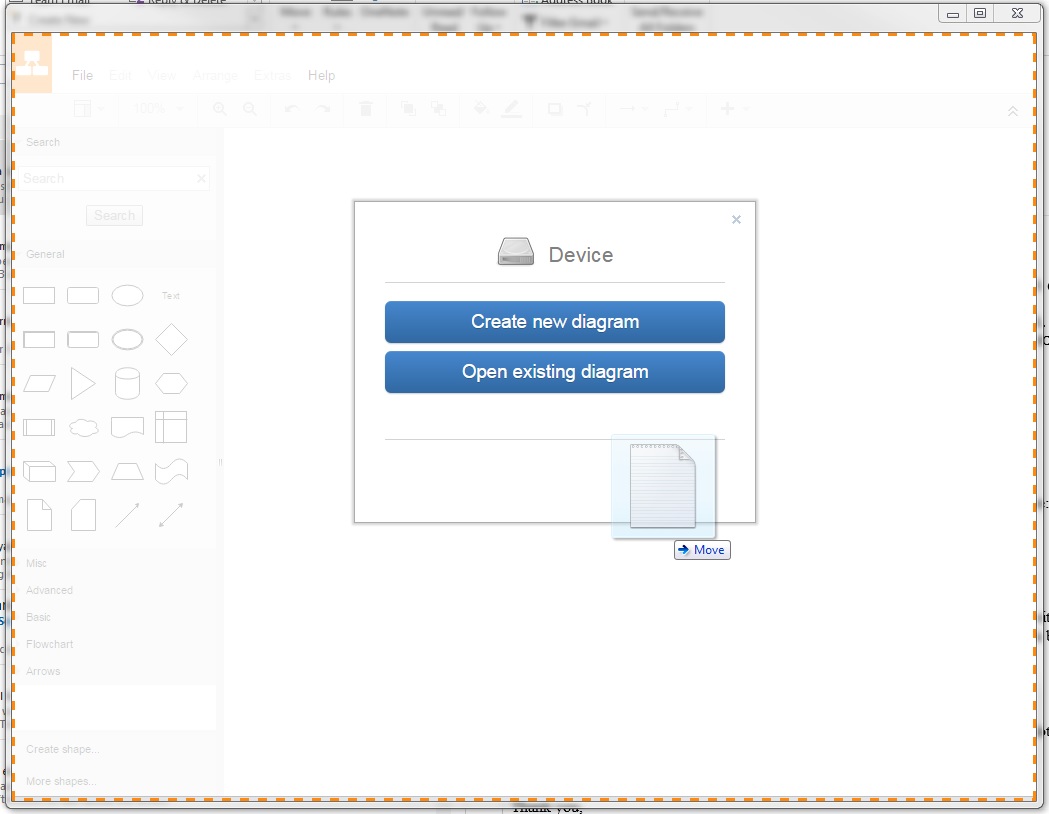


Figure 4 Drag VTK2.0.xml Into Draw.io

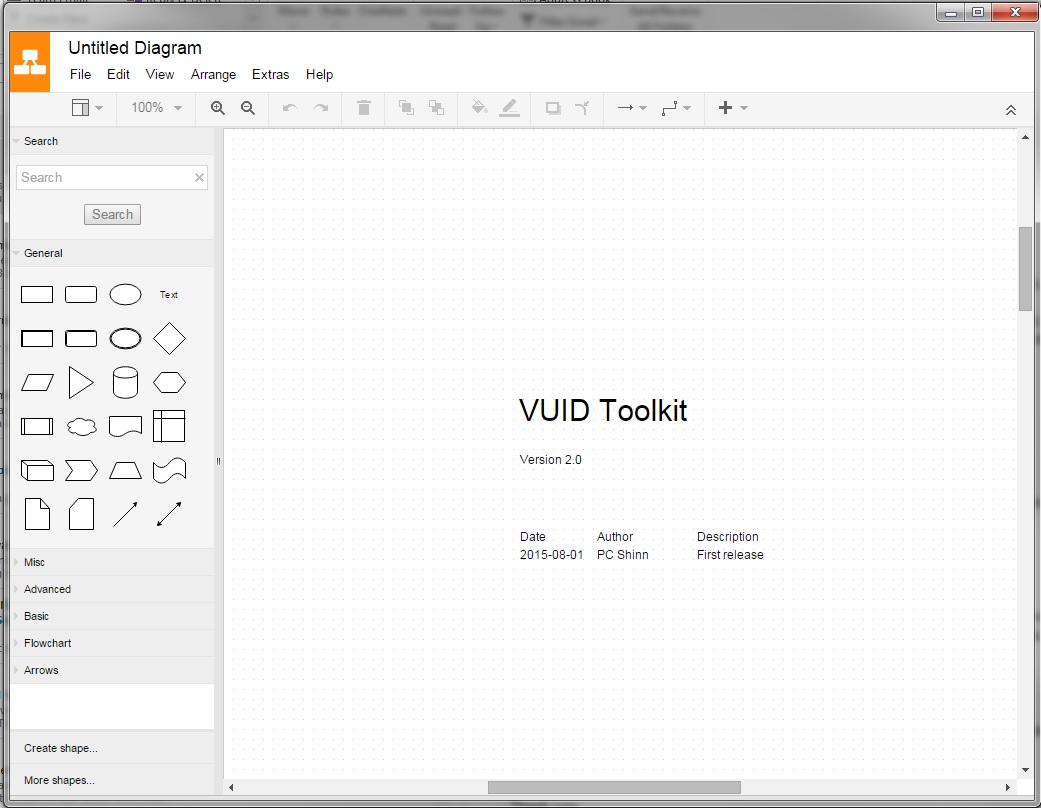


Figure 5 Ready to go!

And you’re ready to start designing.

You might want to save your new design with a new name and not change the original VTK2.0.xml file, since if you break it to start over with a new design you’ll have to download it again.

The first page, or tab, in the template document is used as a cover page with a version number and a list of changes. The second page is a legend that describes the custom objects used by VTK. Take a moment to read this page. The third page is a small sample application.

# Custom Shapes

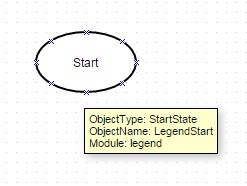
A VTK application is a collection of custom shapes that are used to build a [directed graph](https://en.wikipedia.org/wiki/Directed_graph). More precisely, you can build a [rooted directed graph](https://en.wikipedia.org/wiki/Rooted_graph), since you can tell the application where to start. Since there is also a way to specify where a graph ends, you can group different graphs into modules and hook them up to build sub-routines. More on this later.

There are 8 custom shapes in the toolkit. These shapes are sufficient to design an application, and execute it, but you can create/extend/modify other shapes if you want. Each shape has a property list associated with it. The property list is a list of key-value pairs. To see an object’s property list, hover over it with your mouse (see the yellow boxes below).

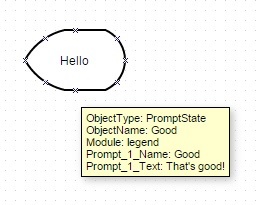
The existing keys and values are used by other parts of the system. You should not change the keys of the existing objects. You should also not change the values of the property ObjectType.

Here is a brief summary of the custom objects.

## Start

For this object, the property ObjectType has the value StartState. You should not change this property or its value. The property ObjectName has the value LegendStart, and you can (and should) change this value depending on the module you are working on. It is best practice to give each unique object a unique ObjectName. The property Module refers to a logical collection of nodes, for example a credit card number collection. There is one start state to a module, and only one outgoing arc from a Start State is allowed. Make sure to change the module property from ‘legend’ to whatever new module you are creating.

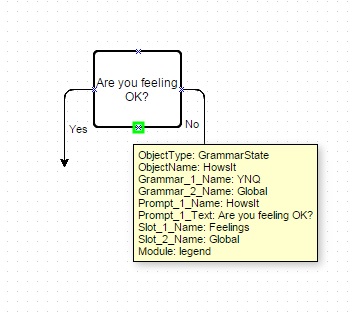
## Prompt State

Use a prompt state to specify what the system says to the user. There can be one or more input arcs, but only one outgoing arc allowed. The ObjectType of PromptState should not be changed. This prompt state has one prompt, whose name is Good and whose text is That’s good! . But you can add as many as you want in the properties list, using the format Prompt\_N\_Name - Prompt\_N\_Text where the prompts are rendered in ascending N order.

## Arc (Arrow)

If there is only one arc exiting a state, it is unlabeled. For grammar states or decision states (see below) the label on the arc represents either what the user said or what the value is of a variable.

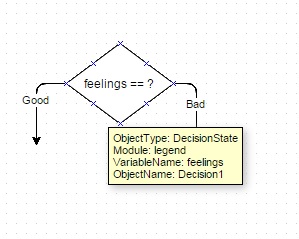
## Grammar State

A grammar state is used to collect input from a user. It is typically designed to output prompts before it does the collection of the user’s input and parses it with a grammar (see below). Labels on the outgoing arcs represent where to go next in the application based on the user's semantic intent. Prompts follow the same conventions as discussed in the prompt state above. Add as many grammars as required. The property list has 2, a state-specific grammar (e.g., YNQ) and a global grammar. Follow the same convention as prompts for additional grammars. Create a grammar object (see below for examples) for every grammar mentioned in the properties list. The slot name in the property list is the field name in a grammar which is populated with a value in the user's response.

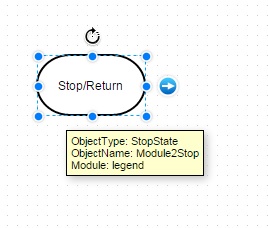
## Computational State

Use a computational state to set variables to values. In this example case, the variable named feelings is being set to the value good. These variables and values can be checked by other states, such as a Decision State (see below).

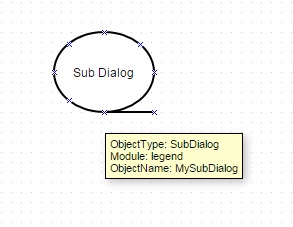
## Decision State

Used to make decisions about where to proceed depending upon the value of the variable. In this case, if the variable named feelings has the value good, then follow the path to the left. If the value is bad, then follow the path to the right.

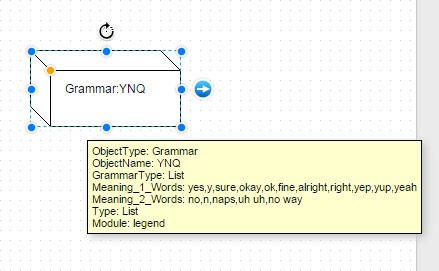
## Stop/Return

Used to end a module or the whole parent application. If the stop state is inside a sub dialog (see below), and there is an arc coming out of the sub dialog symbol, then flow continues back to the state pointed at by the parent sub dialog symbol. If there is no arc coming out of the parent sub dialog symbol, the application stops.

## Sub-Dialog State

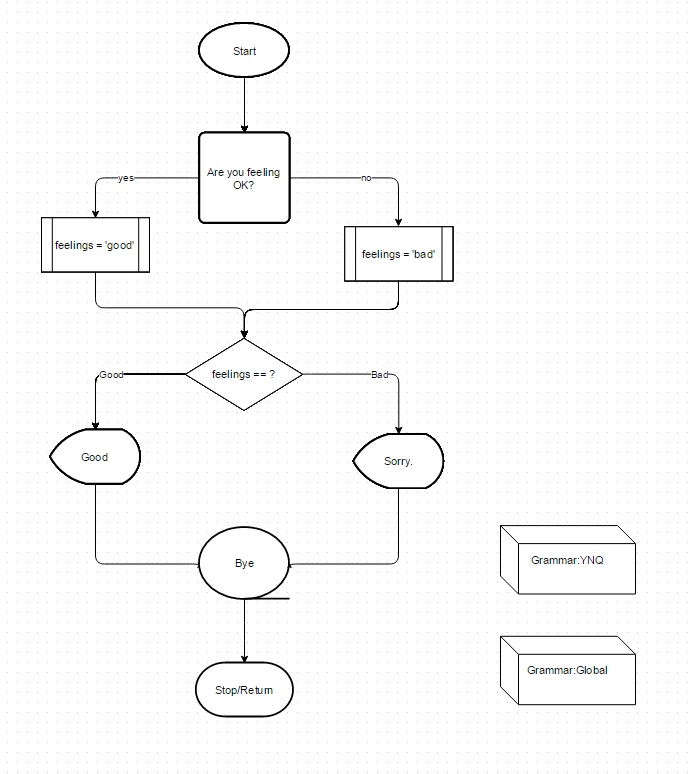
Refers to a separate graph usually existing in another module on a different page. It should have a start state and optionally a return state. By convention, if a Sub Dialog has an exit arc, it acts like a subroutine with a return state. If the Sub Dialog symbol does not have an arc that exits the symbol, than treat it as a labelled Goto, meaning flow of control does not return.

## Grammar

A grammar is used in a Grammar State to parse a user’s input. In this example from the legend, there are two meanings. The first item in the list of words in a meaning is also used as the general semantic intent associated with the meaning. So in this example, the first meaning is yes and the words associated with that meaning are yes, y, sure, okay, fine, etc. The second meaning is no. Every grammar needs to have a property name Type, and in this example, the type of the grammar is List, which is just a simple list of words. Stay tuned for additional grammar types, or feel free to add your own! When this grammar is applied to a user’s input, if the input is yep, then the semantic intent returned to the application by the grammar is yes. The grammar named Global, if present, has special treatment in that if there are two grammars at a state and one of them is named Global, and both grammars have the same terms (i.e., the response from the user is ambiguous) then the default application behavior is to follow the global grammar. You can change this is you want.

# Example Design

Here is the example design in the VTK 1.0.xml file.



One possible interaction is:

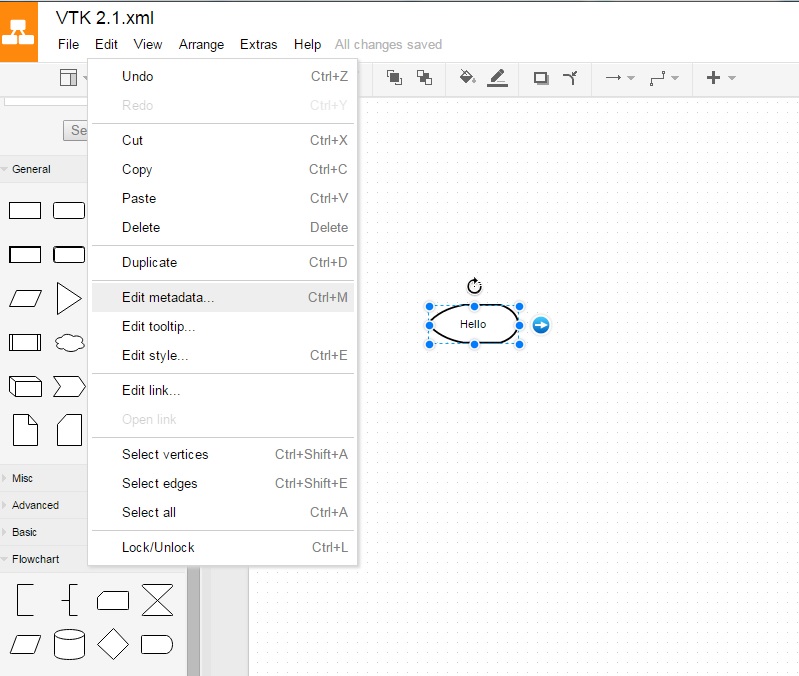
1. System: Are you feeling OK?
2. User: Yes
3. System: That’s good!

The other is:

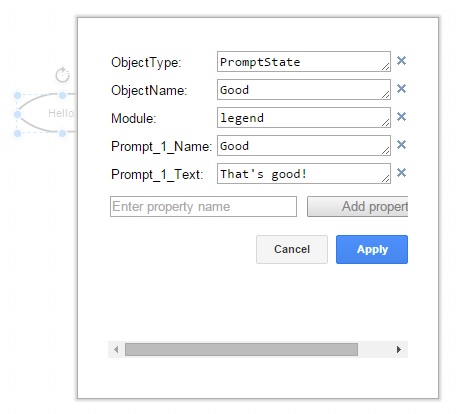
1. System: Are you feeling OK?
2. User: No
3. System: Sorry to hear that.

# Editing/Adding Custom Properties

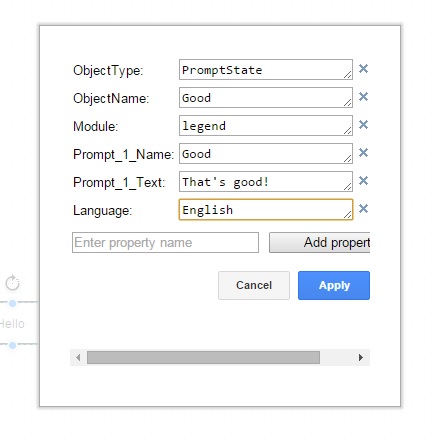
To edit the property list of an object, or to add your own custom properties to the existing objects, select the object, and then from the Edit menu select Edit metadata… :



That brings up the property list:



To add an additional property, in this case Language, fill in a new one in the Enter property name field and select Add property then type in the value. Finish by selecting Apply.



# Sharing a Design

There are a couple of options for sharing a design.

## Export to HTML

From the main menu select File > Export as > HTML. This method exports a picture of the design which cannot be further manipulated within the design tool. But it is a good method for distributing the design for others to review, since all they need to view it is a browser.

## Export to/Import from XML

From the main menu select File > Export as > XML (Plain). Use this method to export the drawing if you want it to shareable with others who want to manipulate the data in the file. This is the method we use to create a starting file for the rest of VTK. There is another option File > Export as > XML (Compressed) which Draw.io uses to save files in native format. You can use this to share designs with others using Draw.io, but the compressed version is not readable by the other tools in VTK. You can then email the exported xml file to a colleague who can import the exported file in Draw.io using File > Import From.

## Cloud Sharing with Draw.io

There are two browser-based cloud versions of Draw.io. One is a single user version at <https://www.draw.io/> which provides the same utility as the desktop version, except that the logic executes in the browser, so no plugin is required. Storage options including Dropbox, Google Drive, OneDrive, the Browser itself or local device.

The other version supports on-line real-time collaboration among multiple users. This version is available at <https://drive.draw.io/> is also known as draw.io Pro which you can access here: <https://chrome.google.com/webstore/detail/drawio-pro/onlkggianjhjenigcpigpjehhpplldkc?hl=en>

Here is a description from the Google Chrome webstore:

**Diagrams Online - The most tightly Google Drive integrated diagramming application available.**

draw.io pro is a diagram editor, built around Google Drive (TM), the technology of which is used in 74 of fortune 100 companies.

draw.io Pro is permanently free for all personal Google accounts.

draw.io focuses on providing enterprise grade:

- Scalability

- Reliability

- Security

- Availability

- Privacy

The key advantages of draw.io for a company basing their cloud storage strategy on Google Drive are:

- All application data is stored only in Google Drive.

- Diagram model is only transmitted directly between browser and Google Drive, server-side data is subject to Google security only.

- draw.io only operates from the https domain, using industry standard SSL encryption.

- Multiple Google accounts are supported, including any number of Google for Work accounts.

- draw.io is hosted on Google App Engine and statically serves a fully operational JavaScript application. The scalability and reliability of draw.io correlate with that of the serving infrastructure only (100% uptime in 2013).

- Your existing backup of Google Drive files will include all draw.io files.

There is some useful information and tutorial videos here on the support page: <https://support.draw.io/display/SUP/draw.io+Online+Support> . These are also hosted on YouTube here: <https://www.youtube.com/watch?v=PDegJXSB2Ho> and here: <https://www.youtube.com/watch?v=TyDqWtNYyrA>

The second tutorial shows how the draw.io.pro version supports real-time simultaneous collaboration, along with chat, and discusses in detail how to share data across the different versions. The on-line version’s manual is here: <https://support.draw.io/display/DO/Draw.io+Online+User+Manual>

One point to take into consideration is they state that these technologies will be free for all existing users from now on for the life of the technologies. So if they do make a decision to start charging for the hosted bits sometime in the future, if you are an existing user at that time they say they will not charge you. Of course if you have downloaded the plug in desktop version then you have it as long as you have the browser and plug in.

1. Currently, the list price of Visio Professional is $590, Visio Standard is $300 and monthly rental is $156 for a year. [↑](#footnote-ref-1)