Gadgeteer VS Designer

Author(s): Carlos Garcia Jurado Suarez

Revision History

|  |  |
| --- | --- |
| 7/20/2011 | Carlos: Initial draft |
| 8/29/2011 | Remove text adornment sections and references |
| 9/1/2011 | Update feature list and added information on build/deploy/sign |
| 9/8/2011 | Added information on releasing drops and full signing |
| 9/12/2011 | Updated build/sign information with designer as part of GadgeteerCore  Added Code overview section and class diagrams |
| 9/28/2011 | Added section on open sourcing the designer |
| 4/5/2012 | Added section on debugging the designer and made minor changes to denote VB is now supported as well |
| 10/6/2015 | Added modeling SDK and common package loading errors |

Contents

[Overview 2](#_Toc321402789)

[Module and Mainboard definitions 2](#_Toc321402790)

[VSModuleManager 3](#_Toc321402791)

[XML Schema 3](#_Toc321402792)

[Designer 4](#_Toc321402793)

[Module and Mainboard end user help 6](#_Toc321402794)

[Logging 6](#_Toc321402795)

[Building 7](#_Toc321402796)

[Debugging 7](#_Toc321402797)

[VS Experimental instance 8](#_Toc321402798)

[VC#/VB Express 9](#_Toc321402799)

[Projects 9](#_Toc321402800)

[Code Overview 10](#_Toc321402801)

[Class hierarchies 10](#_Toc321402802)

[Model Elements 10](#_Toc321402803)

[Presentation Elements 11](#_Toc321402804)

[Hardware Definitions 11](#_Toc321402805)

[Versioning 11](#_Toc321402806)

[Deployment and Signing 11](#_Toc321402807)

[Delay signing for development 11](#_Toc321402808)

[Releasing drops and real signing 12](#_Toc321402809)

[SQM Instrumentation 13](#_Toc321402810)

[Open Sourcing the Designer 14](#_Toc321402811)

[Package Load Key 14](#_Toc321402812)

[Strong Name Signing 14](#_Toc321402813)

[Appendix 1: Class Diagrams 15](#_Toc321402814)

[Model Elements Class Hierarchy 15](#_Toc321402815)

[Presentation Elements Class Hierarchy 16](#_Toc321402816)

[Hardware Definitions Class Hierarchy 17](#_Toc321402817)

# Overview

In addition to .NET Micro Framework templates, Gadgeteer will ship a full Visual Studio designer with code generation capabilities. This will target VC# and VB Express 2010 as well as all editions of Visual Studio 2010 and it is built using the [Visual Studio Modeling and Visualization SDK](http://archive.msdn.microsoft.com/vsvmsdk).

# Module and Mainboard definitions

The designer needs to know about the different modules installed on the user’s computer. Gadgeteer core and third party modules will be deployed through MSIs and they’ll include the module assembly, as well as a GadgeteerHardware.xml file that describes the module’s attributes, requirements and capabilities. In addition to this metadata, the MSIs deploy additional resources used by the designer such as icons or images.

The designer also needs to know about the different mainboard models than can be supported, and this metadata is included in a GadgeteerHardware.xml file as well.

A module installer (MSI) needs to add the following registry keys:

[HKLM\SOFTWARE\**Wow6432Node**\Microsoft\.NETMicroFramework**\v4.1**\AssemblyFoldersEx\*ModuleName*] = *<module assembly installation path>*

[HKLM\SOFTWARE\**Wow6432Node**\Microsoft\.NETGadgeteer\v2\HardwareDefinitionFolders\*ModuleName*] = *<* *GadgeteerHardware.xml directory>*

Where *<module assembly installation path>* is a folder that contains the module assemblies. This is because VS will only look for assemblies in the directory specified in the reg key (not in subfolders). The above example assumes NETMF 4.1, but modules assemblies for different versions, such as 4.2, will need the reg keys to be created under the corresponding AssemblyFoldersEx key.

Note: **Wow6432Node** is only needed on x64 bit systems. MSI will abstract this out as long as the installed is marked as a 32 bit installer.

The first key tells Visual Studio to look here for assemblies. This makes them appear in the Add References dialog and allows Visual Studio to find them at compile time without requiring a full path in the assembly reference. This gives us project portability.

The second key is the mechanism used by the designer to find GadgeteerHardware.xml files.

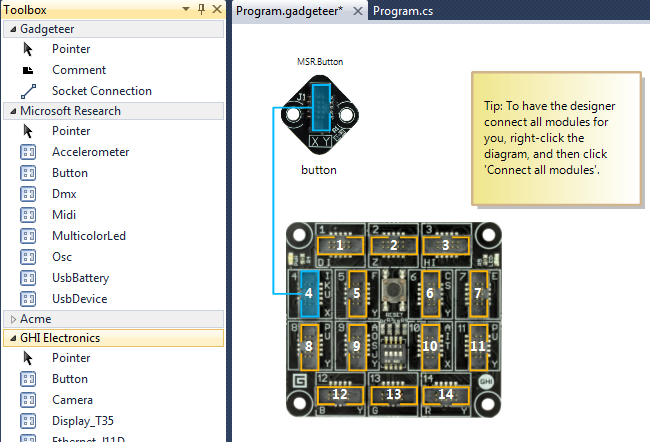
## VSModuleManager

Originally we were going to have additional VS plug-ins using the same metadata from GadgeteerHardware.xml files. Since that may still be the case in the future the module catalog is implemented by a shared component: Microsoft.Gadgeteer.VSModuleManager.dll. This assembly contains the code that looks for valid GadgeteerHardwares.xml files and makes this data available to the adornment and designer.

## XML Schema

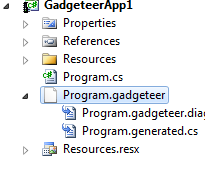
The formal XSD definition is checked in at $/Gadgeteer/Main/GadgeteerCore/GadgeteerHardware.xsd. It contains <xs:annotations> to document the various elements and attributes.

# Designer



The designer is based on the [Visual Studio Modeling and Visualization SDK](http://archive.msdn.microsoft.com/vsvmsdk). The core features are:

1. Show all available modules in the toolbox and allow them to be dragged into the design surface
2. When dropped modules are placed on the surface but not connected. If a reference to the module assembly is not present, it gets added.
3. On each Save or focus change, C# code is generated for:
   1. Program.generated.cs:
      1. Mainboard initialization
      2. Module definitions
      3. Module instantiation and socket connections
   2. Program.cs (user code):
      1. Using statements for the used modules’ namespaces
   3. Here’s the general structure generated:



Where double-clicking on Program.Gadgeteer brings up the designer, double clicking on Program.cs brings up the user code.

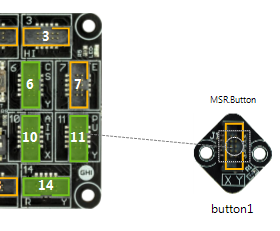
1. Context menus:
   1. Connect all modules: Finds a working configuration. Since modules use different sockets and pins it may be hard for a user with many modules to find a configuration where all modules can be plugged in. We use a simple depth first search algorithm to find one for them.

**Note:** Only sockets not marked as “optional=true” will be used by the solver.

* 1. Go to code
  2. Help/F1 (when a module or mainboard is selected. See [Help](#_Module_and_Mainboard) below)
  3. Disconnect/Del key (on a connected socket)
  4. Cut
  5. Copy
  6. Paste
  7. Properties

1. Modules can be manually connected in two ways:
   1. Using the “module connection tool” from the toolbox. 
   2. Just dragging one socket to another using the selection tool. 

When dragging a connection, compatible sockets light up in green:



1. Certain modules also provide sockets, so we need to support connecting modules to modules. Note that the solver (in V1) is only expected to consider the sockets provider by the mainboard.
2. The user can select different mainboard models (from the property grid when the mainboard is selected).

Out of scope for V1:

1. Using the designer to set module properties or hook up event handlers
2. Reverse engineering existing code into a diagram
3. Round tripping code changes into the designer

# Module and Mainboard end user help

The help to be displayed when the user clicks on the Help menu or when they press F1 (if a module is selected) is determined by the HelpUrl attribute in the corresponding module’s GadgeteerHardware.xml. This url gets launched in a web browser window within VS.

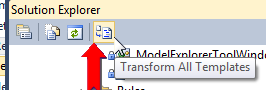
# Logging

The designer will log internal errors and warnings to the [Visual Studio Activity Log](http://msdn.microsoft.com/en-us/library/ms241272.aspx). Internally, the Log class (in Microsoft.Gadgeteer.VSModuleManager.dll) has static methods used throughout the designer to log errors and other information.

# Building

The designer builds as part of GadgeteerCore.sln, although it has an additional dependency on the [Visual Studio Modeling and Visualization SDK](http://archive.msdn.microsoft.com/vsvmsdk).

Occasionally when building, you can see an error saying that **some of the output files were read-only**. This is because the modeling SDK uses T4 code generation and the generated files are in source control. If this happens, click on the Transform all templates button in the solution explorer, which checks-out the necessary files:



Another potential error is when building the designer Merge Module (VisualStudio\Designer\InstallerModule). The reason is that the module is setup to pick the assemblies from the “[signed](#_Releasing_drops_and)” directory and it gets tripped if they are not there. You can temporarily make it build using unsigned assemblies by changing this line in MergeModule.wxs:

    <!--<?define AssemblySrc="$(var.DslPackage.TargetDir)" ?>-->

 <?define AssemblySrc="$(var.DslPackage.ProjectDir)..\..\Signed" ?>

Known issues: From time to time I’ve seen a build error claiming FileTracker.dll can not be found. This is apparently an intermittent build issue with VS and can normally be resolved by clicking the “Transform all templates” button in the solution explorer as described above and/or building the VSDesigner.sln solution from the command line through msbuild.

# Debugging

The project can be launched from VS by making the DslPackage the start-up project. However, if the designer has been installed by the GadgeteerCore MSI (usually the case), then it cannot be uninstalled via VSIX. In this case, visual studio will give this error during build:

"Exception has been thrown by the target of an invocation."

Which is very obscure. The real inner exception (visible under a debugger), shows the actual problem:

System.Exception {System.Reflection.TargetInvocationException}

**\_message "Extension GadgeteerDSL must be uninstalled through the Windows Add/Remove Programs dialog." string**

But you can’t uninstall all of Gadgeteer. So the workaround is to edit the installed vsix manifest, usually at:

c:\Program Files (x86)\Microsoft Visual Studio 10.0\Common7\IDE\Extensions\Microsoft\GadgeteerDSL\1.0.0.0\extension.vsixmanifest

And change the InstalledByMsi element to false:

<InstalledByMsi>**false**</InstalledByMsi>

You can then uninstall the GadgeteerDSL extension from the VS Extension Manager (Options menu)

**IMPORTANT:** VS must be running elevated since it needs to write to %ProgramFiles%.

Once the release build of the designer has been uninstalled, there are a couple of options for debugging:

## VS Experimental instance

This is the default option for VSIX or any VS SDK extension. Visual Studio gets launched with the /rootSuffix Exp parameter which makes it go into an alternate registry location for all the settings. This allows extensions to be registered in isolation from the regular Visual Studio instance. To debug this way, there are a few options that need to be setup in the DslPackage project properties:

**Debug tab:**

Start External program: C:\Program Files (x86)\Microsoft Visual Studio 10.0\Common7\IDE\devenv.exe

Command line arguments: /rootSuffix Exp /Log

The Log option can be useful to debug package loading failures.

**VSIX tab:**



## VC#/VB Express

It is usually more convenient to debug this way because the express skus are much lighter so they launch faster. To enable this, go to the DslPackage project properties:

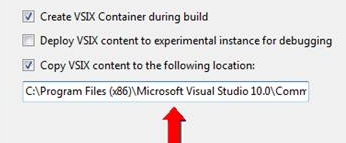
**Debug tab:**

Start External program: C:\Program Files (x86)\Microsoft Visual Studio 10.0\Common7\IDE\**vcsexpress.exe**

Command line arguments: /Log

The Log option can be useful to debug package loading failures.

**VSIX tab:**



The path on my machine is: C:\Program Files (x86)\Microsoft Visual Studio 10.0\Common7\IDE\VCSExpressExtensions\Microsoft\GadgeteerDSL\1.0.0.0

You may need to create the directory if it doesn’t exist.

With either of these two options you should be able to make changes to the code, hit F5 and be debugging your development version of the designer.

**Common errors when loading package:**

**0x80070002**: *File not found.* Use **devenv /log** from Visual Studio command prompt to find out which file is missing (more likely requested incorrectly).

**0x8013141A**: *Strong name validation failed.* When using unsigned assemblies, make sure they are exempted as 32-bit. You can use **sn -Vl** to list the registered assemblies and they should be 3 of them - see below.

# Projects

These are the projects relevant to the designer within GadgeteerCore.sln:

* *VisualStudio*
* *Designer*
  + **Dsl**. The domain specific language with all the semantics for the designer including the meta-model, shapes, rules, etc.
  + **DslPackage**. VS Package implementation for the designer. This hosts most of the VS integration, suchs as the DocData, DocView, Commands and code generation.
  + **GadgeteerDesignerUnitTests**. Regular VS unit test project. The tests cover the main algorithms in the designer (such as socket matching and the constraint solver)
  + **InstallerModule**. WiX merge module wrapping the installer installation
  + **Setup**. WiX installer that simply wraps the above merge module.
* **VSModuleManager**. Module catalog manager implementation and common classes (such as logging) used by the designer.

# Code Overview

As mentioned before, the code is based on the Visual Studio Modeling and Visualization SDK, so its important to become familiar with this framework to understand the code. Here are the main elements of the designer:

1. **DSL definition**. The definition of the designer domain specific language. This is contained in Dsl\DslDefinition.dsl. This file is the input to a series of text templates that generate the code for the designer.
2. **Generated code**. See under Dsl\GeneratedCode and DslPackage\GeneratedCode. The .tt files generate the corresponding .cs files.
3. **Customizations**. The various classes generated by the text templates are [derived or double derived partial classes](http://msdn.microsoft.com/en-us/library/bb126289.aspx). These are the further customized through the code in Dsl\CustomCode and DslPackage\CustomCode.
4. **Rules**. See Dsl\Rules\\* and <http://msdn.microsoft.com/en-us/library/bb126258.aspx>.
5. **Events.** Used to implement the assembly referencing in DslPackage\CustomCode\GadgeteerDslDocData.cs. See also <http://msdn.microsoft.com/en-us/library/bb126250.aspx>.
6. **Context menu commands**. Defined in DslPackage\Commands.vsct and implemented in DslPackage\CustomCode\CommandSet.cs.

## Class hierarchies

The designer code base contains 3 class hierarchies:

### Model Elements

*Microsoft.Gadgeteer.Designer namespace*

These represent instances of Gadgeteer hardware in a model. Examples are Mainboards and SocketUse. They encapsulate the semantics of how the hardware fits together. See [Model Elements Class Hierarchy](#_Model_Elements_Class).

### Presentation Elements

*Microsoft.Gadgeteer.Designer namespace*

Presentation elements are the visual representation of the model elements on the design surface. Examples are: GadgeteerHardwareShape and SocketShape. See [Presentation Elements Class Hierarchy](#_Presentation_Elements_Class).

### Hardware Definitions

*Microsoft.Gadgeteer.Designer.Definitions namespace*

Definition objects are deserialized from the GadgeteerHardware.xml files and represent the metadata that defines the specific hardware. For example, a general purpose Module model element has a pointer to the specific module definition, such as an SDCard ModuleDefinition. See [Hardware Definitions Class Hierarchy](#_Hardware_Definitions_Class).

# Versioning

In order to maintain a single version for the assemblies, the VSIX package and the installer, the versioning is done via T4 templates that read from a common file. To change the version number, edit

**$Main\GadgeteerCore\VisualStudio\version.txt**

And then click the “transform all templates” button described above. This file is used as an input to T4 templates that generate these files:

* AssemblyVersionInfo.cs
* Designer\DslPackage\source.extension.vsixmanifest
* Designer\DslPackage\WixVersion.wxi

This keeps versioning consistent across the assemblies, the VSIX and the MSI. Remember to check these files in when releasing drops.

# Deployment and Signing

## Delay signing for development

Designer and adornment assemblies need to be Strong Name signed using the Microsoft key. During testing and in development builds these will be delay signed with just the public part of the key. For this reason signature validation needs to be disabled on dev/test machines if using and unsigned build:

C:\Program Files (x86)\Microsoft SDKs\Windows\v7.0A\Bin\NETFX 4.0 Tools\sn.exe

From an **administrator** prompt run:

sn -Vr <dll name>

For these dlls:

Microsoft.Gadgeteer.Designer.DslPackage.dll  
Microsoft.Gadgeteer.Designer.Dsl.dll  
Microsoft.Gadgeteer.VSModuleManager.dll

**Note**: On x64 run the x86 version of ‘sn.exe’. Using ‘Visual Studio x64 Win64 Command Prompt’ will edit the wrong registery values and result in strong name errors during build.

Here are a couple of good links that explain what strong name signing is and why we have to do the “sn –Vr …” steps.

<http://msdn.microsoft.com/en-us/library/h4fa028b(v=VS.100).aspx>   
<http://msdn.microsoft.com/en-us/library/t07a3dye(VS.80).aspx>

## Releasing drops and real signing

For actual releases the assemblies need to be properly signed. If this is being done for the first time, follow [these](http://codesigninfo/Wiki/Getting%20Started.aspx) instructions to configure the required software and account permissions. Then:

1. Increment the version number as described in [Versioning](#_Versioning)
2. Open a command prompt
3. Cd to \Main\GadgeteerCore\VisualStudio
4. Run:

BuildAndSign.cmd

This takes a few seconds and then will ask for your smartcard and pin to submit the assemblies for signing. The list of approvers is currently hardcoded inside the SignAssemblies.vbs script, so edit accordingly.

1. Once you get an email about signing being complete, Run:

ReleaseBuild.cmd <signing output share> (this comes in the email)

For example

ReleaseBuild.cmd [\\PR1CS2CFILE1.redmond.corp.microsoft.com\02\CMP\CGARCIA\CSD10654537](file:///\\PR1CS2CFILE1.redmond.corp.microsoft.com\02\CMP\CGARCIA\CSD10654537)

1. You’ll get a prompt for checking in the build along with the symbols. Just click “Check in”
2. Check-in the files changed due to the version update:
   1. AssemblyVersionInfo.cs
   2. version.txt
   3. Designer\DslPackage\source.extension.vsixmanifest
   4. Designer\DslPackage\WixVersion.wxi

The resulting MSI will be in $Main/Builds/Development Builds/GadgeteerCore-<version>.msi.

# SQM Instrumentation

Visual Studio has built-in support for reporting SQM data so we get this for free. VS will log data around our template instantiations, designer and commands use. We need to ask the VS SQM team to register the various designer components as ours to get sensible reports (this is in progress).

**From the VS team:**

To look at Add new item data, go to [SQMExplorer](http://sqmexplorer/SqmExplorervisualstudio/default.aspx), on the upper portion of the window will be a toolbar. Click on cid:image001.png@01CC626A.88A98C10, go to the [\\vspfs1\public\dazitzel\SqmReports](file:///\\vspfs1\public\dazitzel\SqmReports) folder, and open RecentTemplates.sqmx.

Click the Green “Run” arrow and then look for your templates in the list.

Following the instructions for <http://devdiv/sites/vspro/telemetry/Telemetry%20Wiki/Testing%20your%20SQM%20data.aspx> will allow you to determine if your templates are defined or not.

If your templates are reporting unknown, then you need to work with Selma to get your templates defined.

Once they are defined, you can usually start seeing data about 2 weeks afterwards.

Our current contacts are David Zitzelsberger [dazitzel@microsoft.com](mailto:dazitzel@microsoft.com) ; Curt Becker [curtbe@microsoft.com](mailto:curtbe@microsoft.com) ; Selma Ikiz [selmai@microsoft.com](mailto:selmai@microsoft.com); Radames Cruz Moreno [Radames.Cruz@microsoft.com](mailto:Radames.Cruz@microsoft.com)

# Open Sourcing the Designer

There are a couple of logistical issues to address if the designer code is published as open source:

## Package Load Key

C#/VB Express is designed to not allow VSIX extensions other than templates. So in order to get the designer to load in Express, the VS team generated a Package Load Key for us. If we open source the designer code, we need to make sure this key is not published in the source. This means people that build the designer on their own will not be able to load it inside Express SKUs, but it’ll work in VS.

The PLK resides in ***Main\GadgeteerCore\VisualStudio\Designer\DslPackage\VSPackage.resx***, as resource id **105**. It’s a signed hash of the designer’s name and version, so changing any og this will invalidate the key. This key gets applied to the package through the following attribute in Main\GadgeteerCore\VisualStudio\Designer\DslPackage\GeneratedCode\Package.tt (which in turn generates Package.cs):

**[VSShell::ProvideLoadKey("Standard", "1.0.0.0", "GadgeteerDSL", "Microsoft", 105)]** internal sealed partial class <#= dslName #>Package : <#= dslName #>PackageBase  
 {}

## Strong Name Signing

Our designer [projects](#_Projects) are delay signed with a Microsoft key. If they are open sourced, this project setting needs to be removed since nobody outside of MS can release sign with out certs.

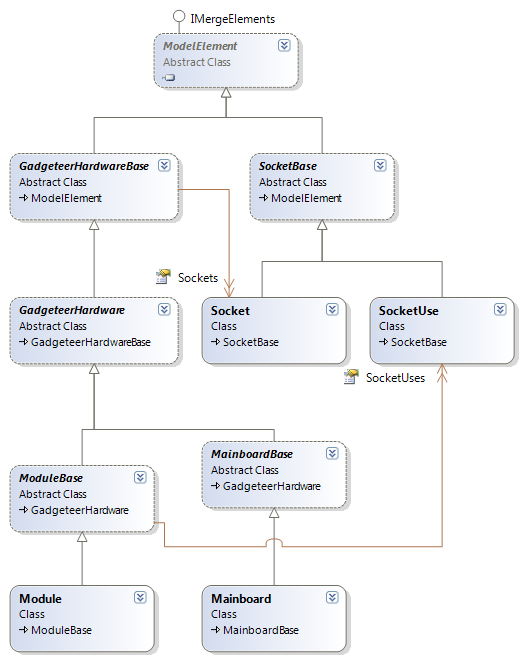
## Modelling SDK

In order to allow building for all supported Visual Studio versions without the need to have those versions installed, the SDK binaries are cached in ***Main\GadgeteerCore\VisualStudio\Designer\ModelingCache***. These are probably not redistributable and shouldn’t be part of the repository.

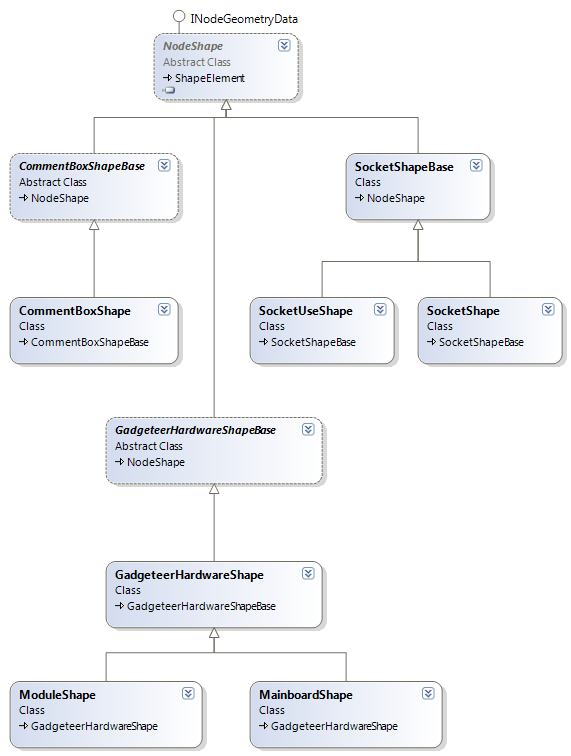
# 

# Appendix 1: Class Diagrams

## Model Elements Class Hierarchy



## Presentation Elements Class Hierarchy



## Hardware Definitions Class Hierarchy

