WWW Script Generator

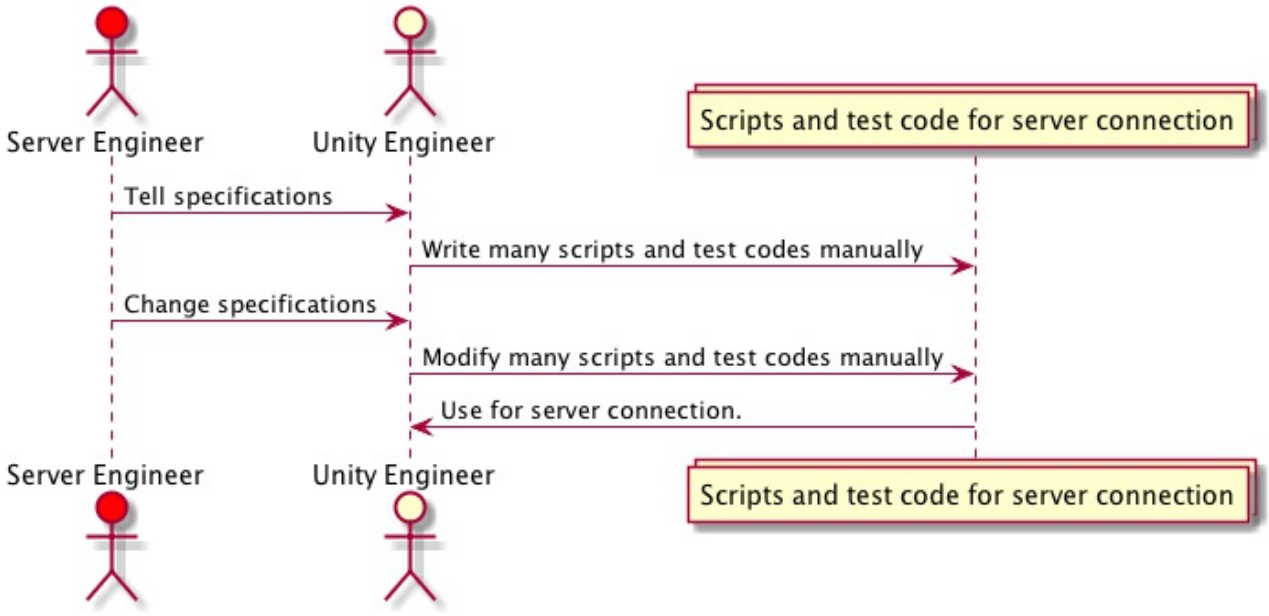
Change log

2018.2.9

- Add 'Handle Cookie' and 'PlayerPrefs key for Cookie' to the basic setting screen
- Added a description of how to specify the position of the parameter in the 'Endpoint' of the method screen.
- Added descriptions of 'Class variables of base class', 'Class method of base class','Class method of APITask' in 'Customizing APITask'
- Added supplement for 'CreateHeader' in 'Customizing APITask'

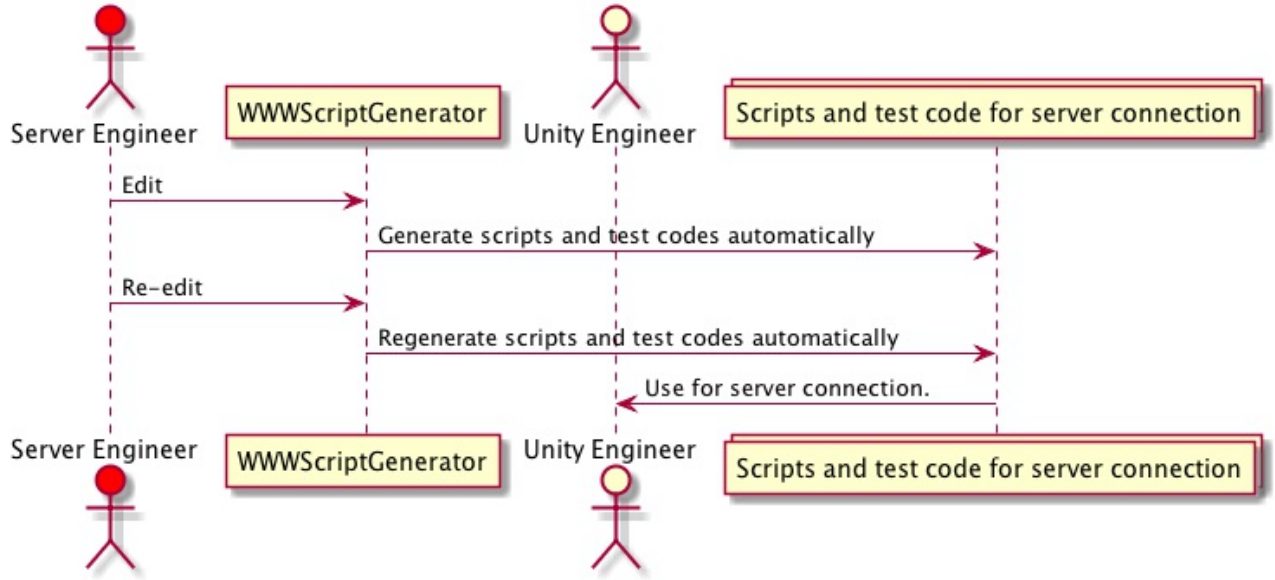
About “WWW Script Generator”

Connection with the server is becoming essential in recent game development. However, Unity developers and server developers are often different people, there will be a lot of communication about specifications, and communication mistakes may occur.
Also, if the number of APIs is large, there are many scripts that must be created for that, and coding takes a huge amount of time.
In many cases, development starts at the stage where the specification is not settled, so specification changes occur frequently, and each time, Unity developers have to modify the script while keeping track of the changes.
Moreover, it requires a lot of man-hours for connection test, too.



With this package, you can eliminate specification mistransmission by sharing specifications in setting files with server developers.
Also, the automatic script generation function greatly shortens the time required for script creation.
Even when the specification changes, you can easily respond by clearly grasping the changed part and regenerating the script.

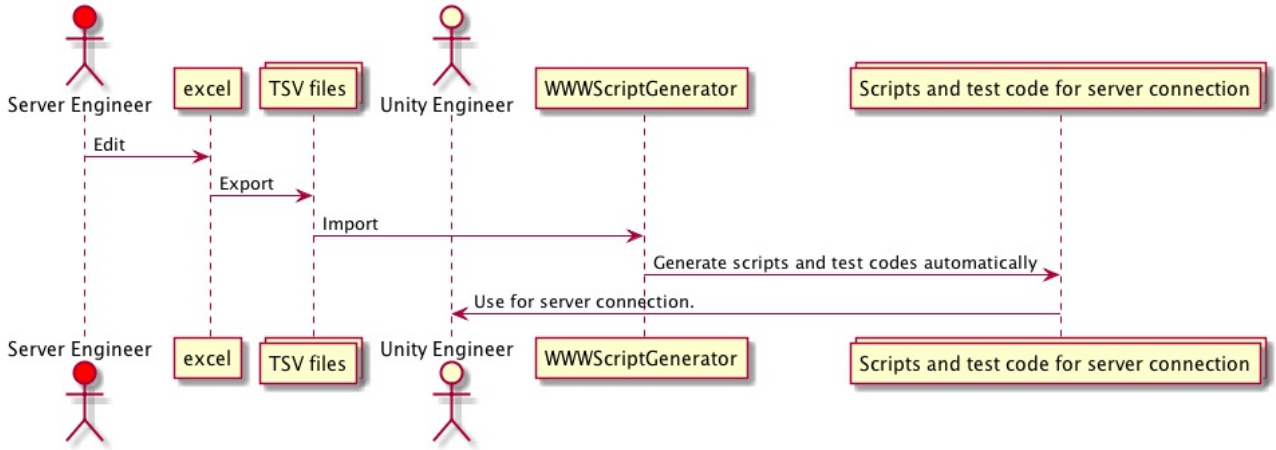
And by using the automatically generated test code, you can greatly reduce the time required for communication test.



Since the automatically generated script is specialized in the server API, it can be clearly separated from other logic and readability is improved.
The class used for connection is Unity standard “WWW”, we do not use the own function of this package. The generated script does not depend on the dll of the package, and you can customize the generated class and make various adjustment such as the header information and format of the transmitted data to the server, so it can handle any connection.

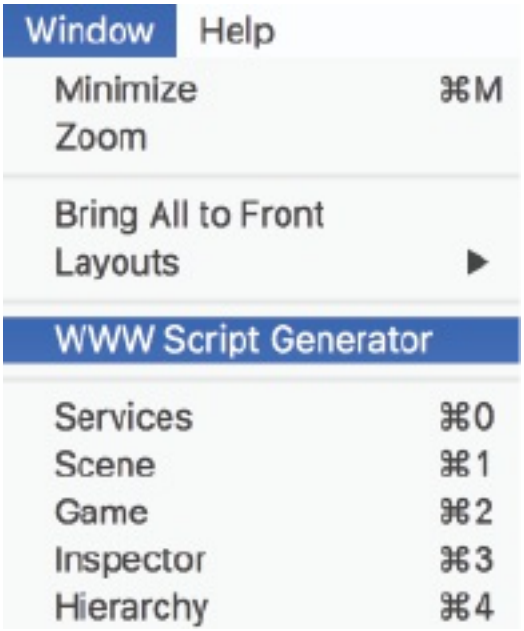
Since model classes for parsing JSON are also automatically generated, JSON can be handled more safely than using JSON as it is.

Furthermore, TSV file can be read and written in the Pro version, it can link with external tools such as Excel, and it will be possible to link with server developers who do not have Unity.

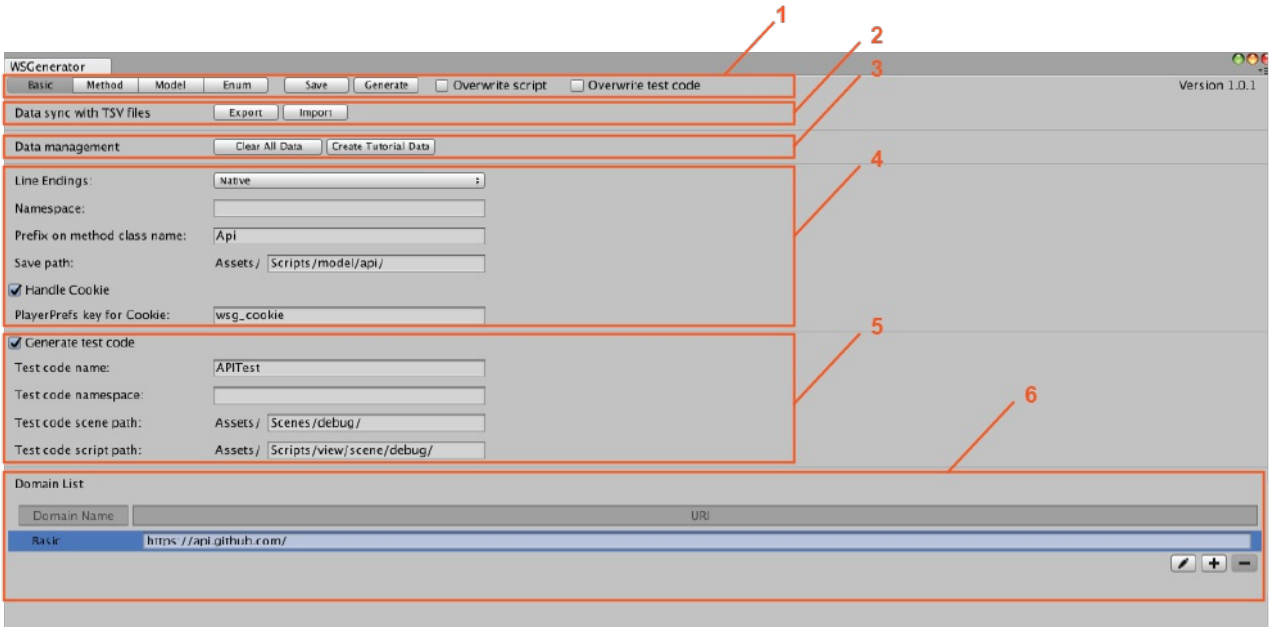


Screen description

First of all, choose *"Window / WWW Script Generator"* from the menu of Unity and open the window.

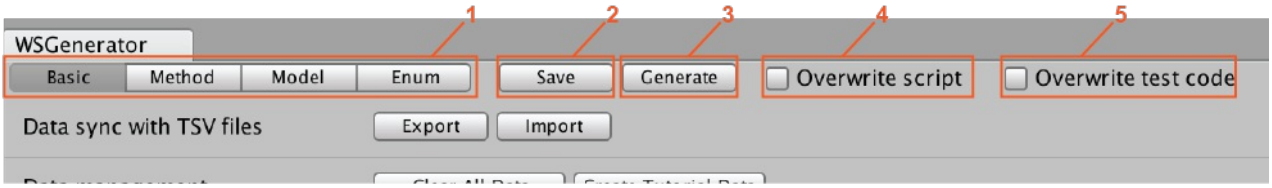


Basic (Basic setting)



1. Common Menu: Common menu for each screen
2. Data sharing setting: Import / export setting file as TSV (※ This function is limited to Pro version.)
3. Data management: Data clear and tutorial data creation.
4. Code Generation / Behaviour Setting: Setting for Code generation and behaviour
5. Test code generation setting: Setting related to test code generation
6. Domain list: Define the URI of the connection destination

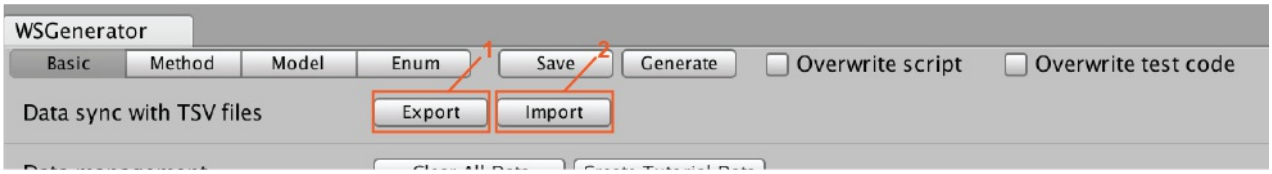
Common menu



1. Screen change tab
    - Basic: Switch to basic setting
    - Method: Switch to method list
    - Model: Switch to model list
    - Enum: Switch to enum list
2. Save: save setting button
3. Generate: Script generation button
4. Overwrite script: Forcibly overwrite cs files to customize when generating scripts.
5. Overwrite test code: Forcibly overwrite cs files to customize when generating test code.

※ The file for customize is described in “Generating script / test code”

Data sharing setting



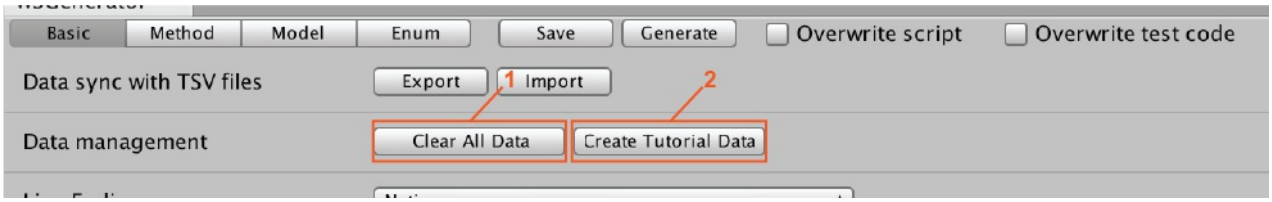
1. Export: Export the setting file as TSV.
2. Import: Load TSV file.

※ This function is limited to Pro version.

Each TSV file is equivalent to the contents of the following screen.

- wsg_domain.tsv: Basic setting of Domain List
- wsg_common.tsv: Common parameters of the method list
- wsg_method.tsv: Method list and parameters
- wsg_model.tsv: Model list and properties
- wsg_enum.tsv: enum list and values

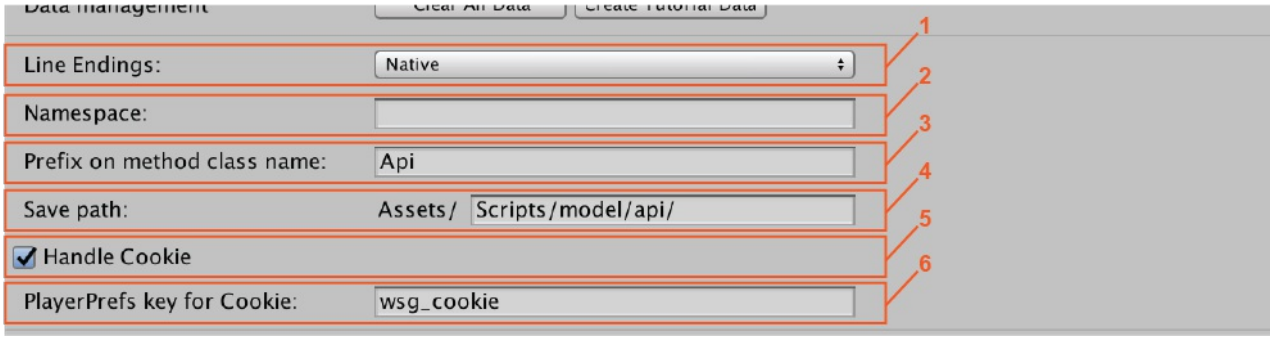
Data management



1. Clear All Data: Clear all data which user input.
2. Create Tutorial Data: Create data for tutorial.

※ Tutorial data (temporary data of method and model) is best for first-time users to learn how to use them.

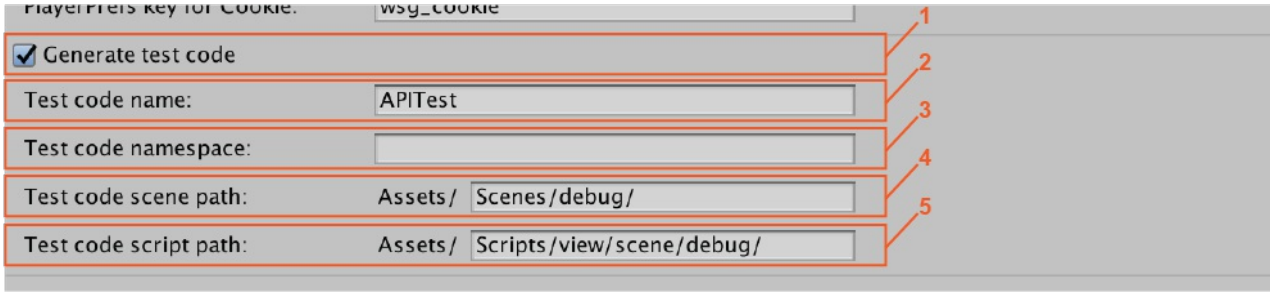
Code Generation / Behaviour Setting:



1. Line Endings: Specify the line feed code of the script to be exported.
    - Native: It matches the line feed code of OS
    - Mac Classic: \r(CR): Old Mac OS (9 or earlier)
    - Unix Mac: \n(LF): Unix, Mac OS X
    - Microsoft Windows: \r\n(CR+LF): Windows

2. Namespace: Specify the namespace to be added to the script to be exported. If you do not specify a namespace, leave it blank.
3. Prefix on method class name: You can specify the prefix to be added to the class name of the method.
   If there are a lot of methods, specify a prefix to make it easier to find and distinguish it from other classes the time of coding.
   Leave it blank if not specified.
4. Save path: Specify the directory where you want to save the generated script.
5. Whether to handle cookie or not. Save the cookie contained in the header of the received data in PlayerPrefs and include it in the header of the transmitted data."
6. Key name for saving cookie in PlayerPrefs

Test code generation setting



1. Create test code: Specify whether to generate test code at the same time as generating

script.
2. Test code name: Specify the test code file name (scene name, script name).
3. Test code namespace: Specify the namespace to be added to the test code. If you do not specify a namespace, leave it blank.
4. Test code scene path: Specify the directory where the scene file of the test code is saved.
5. Test code script path: Specify the directory where the test code script file is saved.

Domain list

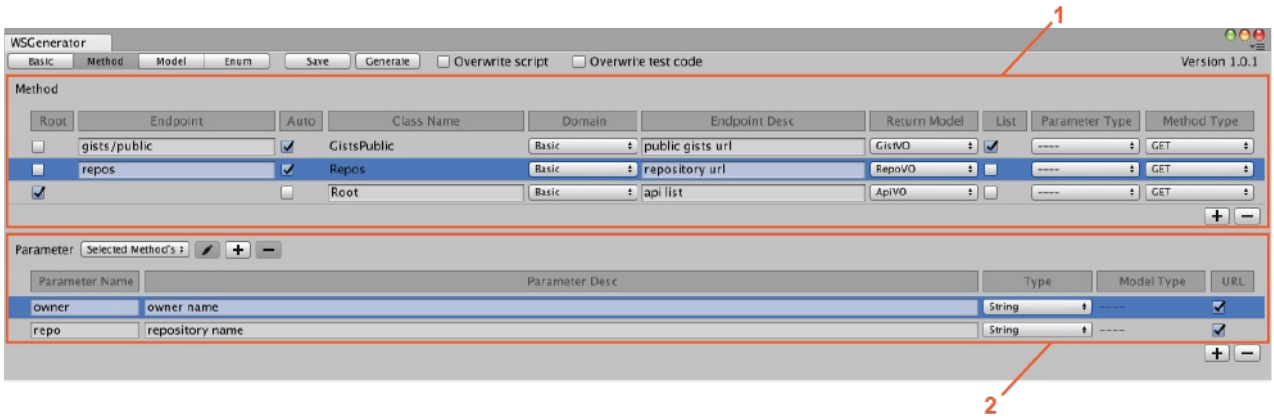
Specify the server connection destination. If there are multiple connection destinations, add items with the + button.

1. Domain Name: Name of the domain
2. URI: domain URI
3. Edit the name of the domain
4. Increase items
5. Delete the selected item

Method list

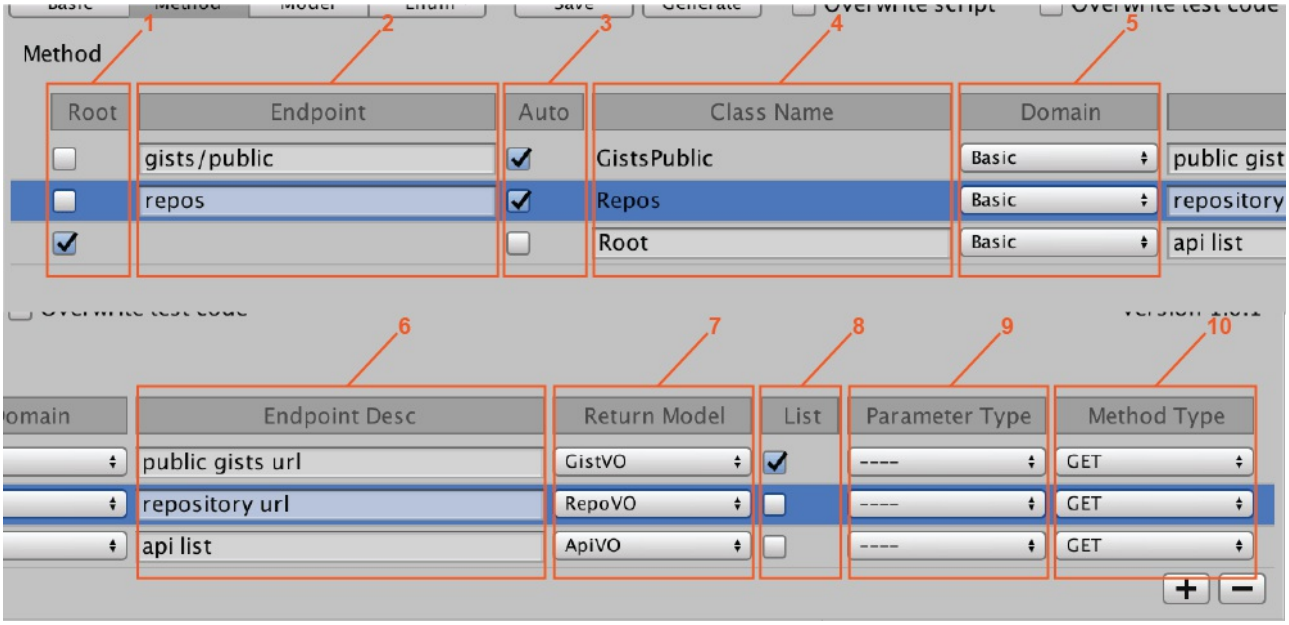
For using the generated method class, instead of instantiating, call the Access method of the class method. For example, when calling a method called “ApiScorelist”, you can call it as follows.

```
ApiScorelist.Access(memberId, (vo, rawText) =>
{
    Debug.Log("Success " + vo.ToString());
}
,
(APIError err, string rawText)
{
    Debug.Log("Error " + err.Code + ", " + err.Message);
}, gameObject);
```



1. Method: List of methods
2. Parameter: Method parameters

List of methods

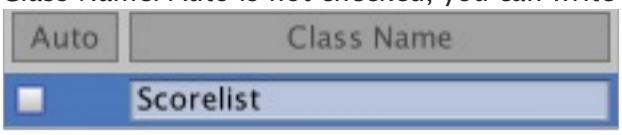


1. Root: Check this if you do not have an endpoint and you want to use the URI specified in the Domain list as it is.

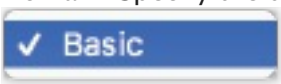
2. Endpoint: The endpoint of the method. Write a string following the URI specified in the Domain list. If a parameter is contain in the endpoint, put a check in 'URL' of the parameter, enclose the parameter name with {} and put it in the endpoint. For example, if 'user/getdata/' followed by a parameter named 'user_id' is entered, the endpoint will be 'user/getdata/{user_id}'. Even if you omit the parameter and leave it as 'user/getdata', if the 'URL' is checked, the parameter will be inserted after the endpoint

3. Auto: When this is checked, the class name of this method is automatically created from the endpoint. Extensions such as “.json” and “.php” are not included in automatically generated names.

4. Class Name: Auto is not checked, you can write the class name of this method manually.

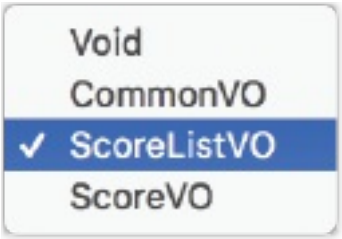


5. Domain: Specify the domain URI to use from the domain list.



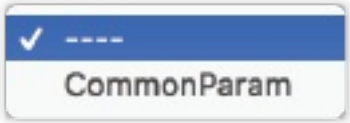
6. Endpoint Desc: It is for describing what purpose this endpoint is, it will be a comment in the generated script.

7. Return Model: Specify the return value of method from models. If there is no return value, select “Void”.



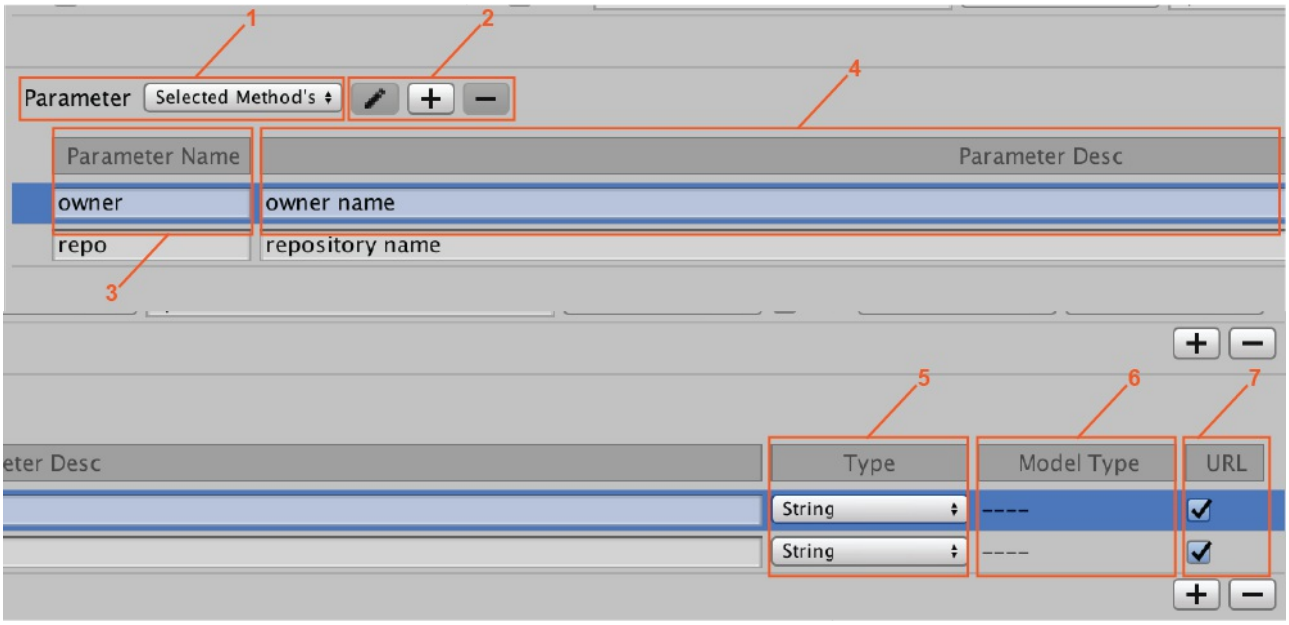
8. List: It is specified when the return value of JSON is an array.
   For example, if JSON is
   `[{"name":"a"}, {"name":"b"}]`
   instead of
   `{"list":[{"name":"a"}, {"name":"b"}]}`
   , check this.

9. Parameter Type: Specify common parameters. If you do not specify it, choose “----”.

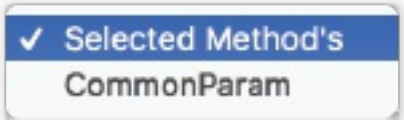


10. Method Type: Choose either POST or GET for connection method.

Method parameters



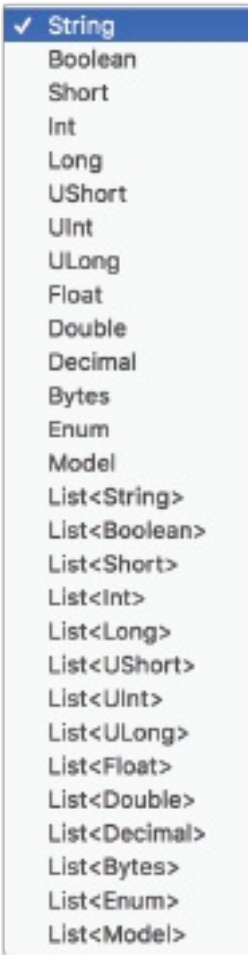
1. Choose between parameters for the currently selected method or common parameters.



   - Selected Method's: Parameters of the currently selected method

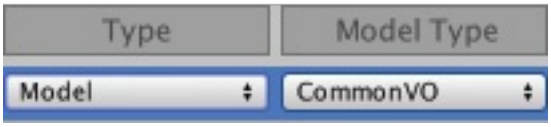
    - Other: Common parameters

2. Edit, add, and delete common parameters.
3. Parameter Name: Specify the parameter name. It is the same as the key name sent to the server when actually communicating with the server.
4. Parameter Desc: It is for describing the parameter, it will be a comment in the generated script.
5. Type: Specify the type of the parameter.



6. Model Type: Choose which type to use when the type is `Model`, `Enum`, `List<Model>`, `List<Enum>`.
   For `Model`, `List<Model>`, choose from Model.

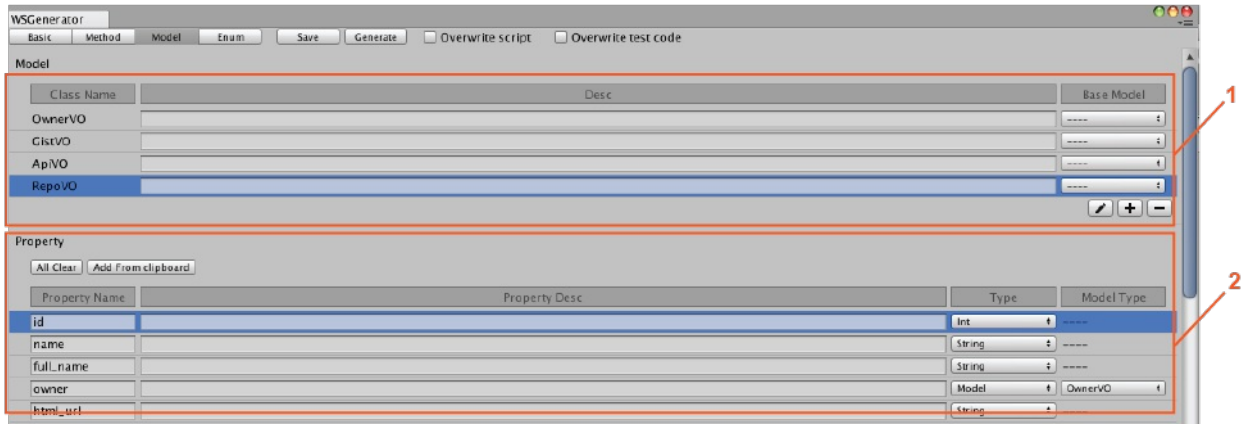
   For `Enum`, `List<Enum>`, choose from Enum.



7. URL: Whether the parameter is in the URL
   For example, if the value of the parameter "param1" is "value1", the actual URL is `https://example.com/value1`. If the endpoint contains a parameter name which is enclosed in {}, the value of parameter will be inserted that position.

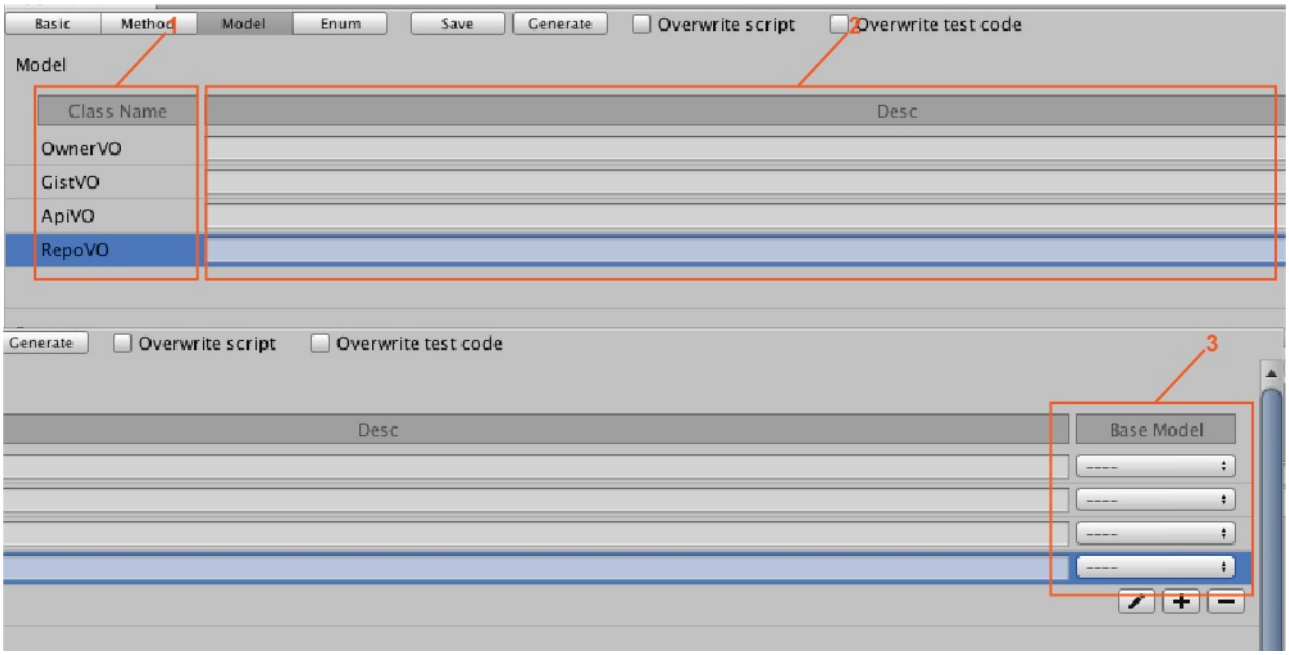
Model list

Here we define the model used for method parameters, method return values, etc. “VO” at the end of the model name is an abbreviation for “Value Object”, which means that it is an object for storing values without logic. A model can have other models as properties.



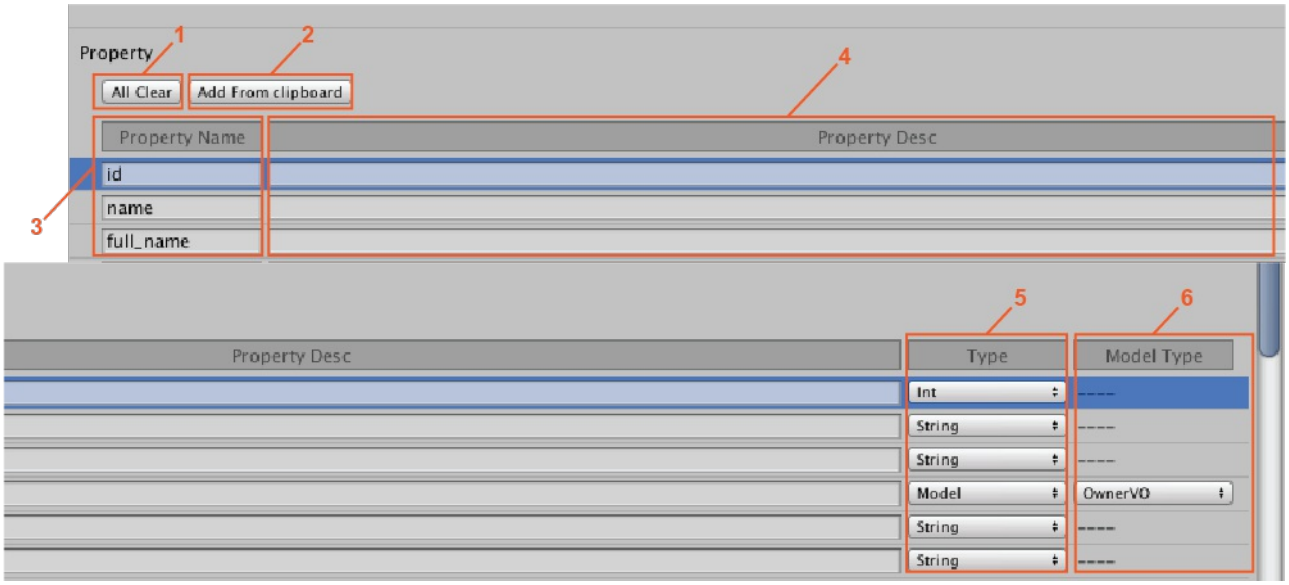
1. List of models
2. Model properties

List of models



1. Class Name: Specify the class name of the model.
2. Desc: It is for descripting the model, it will be a comment in the generated script.
3. Base Model: If there is a model with common properties, you can create a common class and inherit it. If you do not specify it, choose “----”.

Model properties



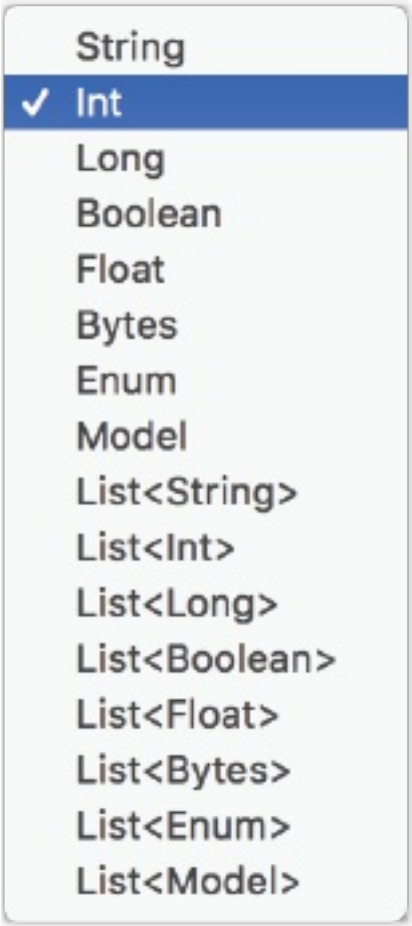
1. All Clear: Delete all properties of the selected model.

2. Add From clipbboard: Create properties from the JSON string in the clipboard.
   It is useful if there is already JSON of return value from the server. For example, if you press this button with the state clipboard contains the JSON character string `{"key1":"value1", "key2":"value2"}`, "key1" and "key2" will be added.
   It is OK even if it is not surrounded by "{" and "}", like `"key1":"value1", "key2":"value2"`.

3. Property Name: Specify the property name. Make it the same as the JSON key name when actually communicating with the server.
   ※ The property name is userd as a member variable of the C# class in the generated script. So reserved words can not be used as property names.
   (E.g. public, protected, private)

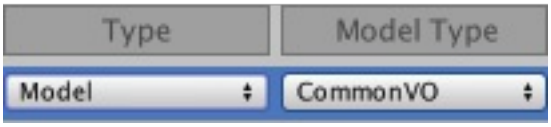
4. Property Desc: It is for describing the property, it will be a comment in the generated script.

5. Type: Specify the type of the parameter.



6. Model Type: Choose which type to use when the type is `Model`, `Enum`, `List<Model>`, `List<Enum>`.
   For `Model`, `List<Model>`, choose from Model.

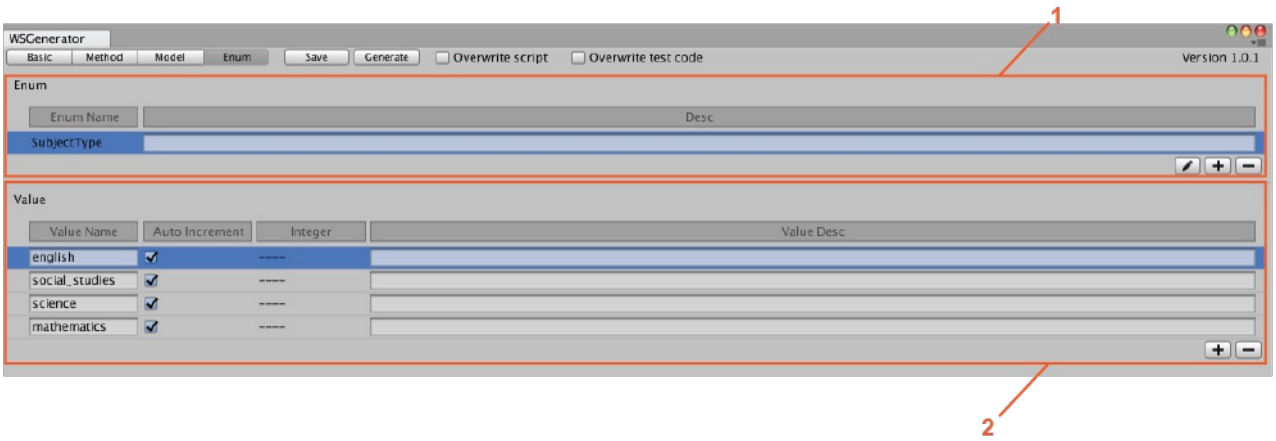
   For `Enum`, `List<Enum>`, choose from Enum.



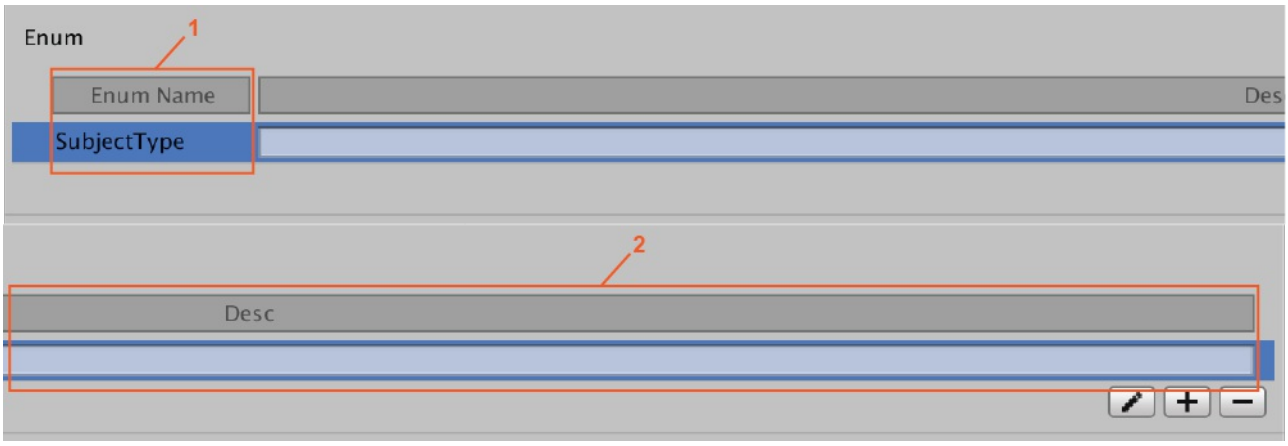
※ You can specify “List” as the type of property, but not “Dictionary”. WWWScriptGenerator uses JsonUtility to parse from JSON to a model, but the Dictionary type is not supported in JsonUtility. If you want to make it a property of type Dictionary, you can manually convert it to a Dictionary after parsing it as List.

enum list

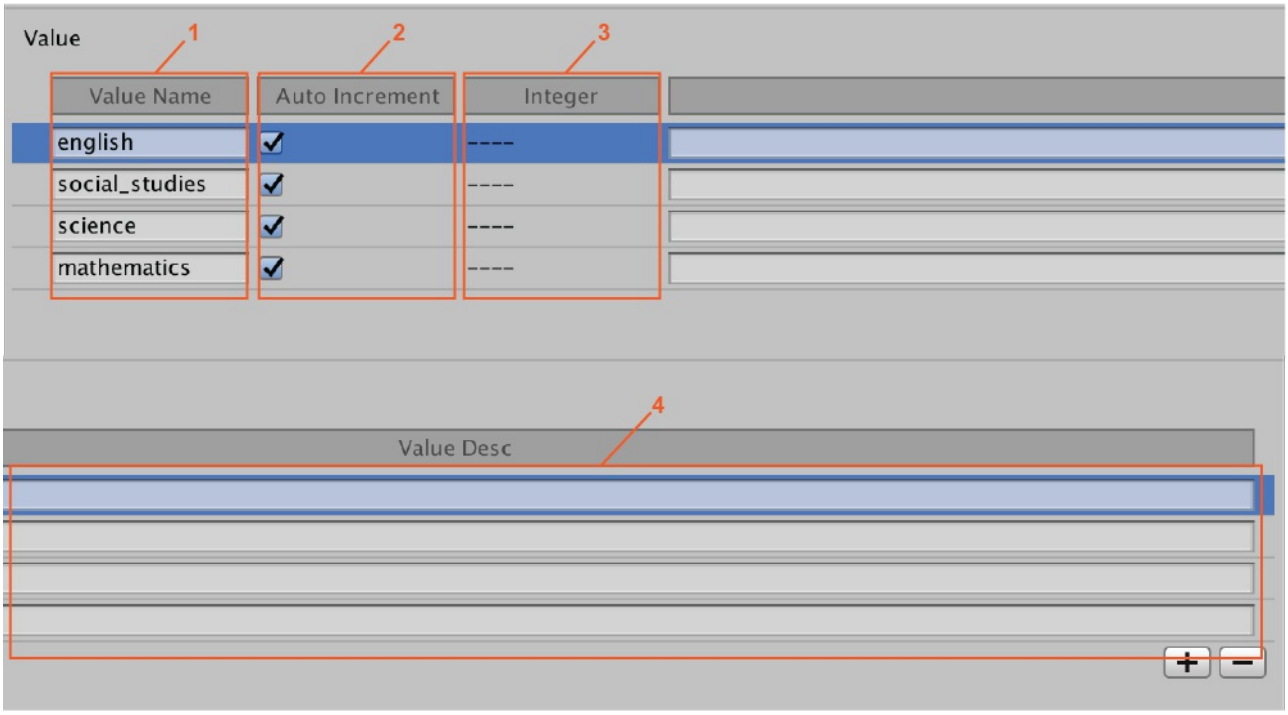
Here we define the enums to use for method arguments and model parameters. For example, if there is a parameter of “Subject” whose value 0 means English and 1 means Social studies, enum is more readable than directly handling as integer.
Integer usually starts from 0 and increases by 1, but you can specify unique numbers such as 1000 for English and 2000 for Social studies.



1. List of enum
2. Enum values

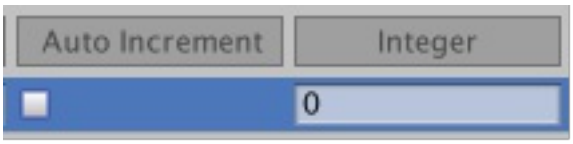


1. Enum Name: Specify the name of enum
2. Desc: It is for descripting the enum, it will be a comment in the generated script.



1. Value Name: Specify the name of the enum's value. Since it is converted to Pascal in the exported code, it does not have to be capitalized here.

2. Auto Increment: Set False if you want to specify an integer value. If True, the value of the integer is the previous value plus one.
3. Integer: If you want to specify an integer value, set Auto Increment to False and specify a value.



4. Value Desc: It is for describing the enum's value, it will be a comment in the generated script.

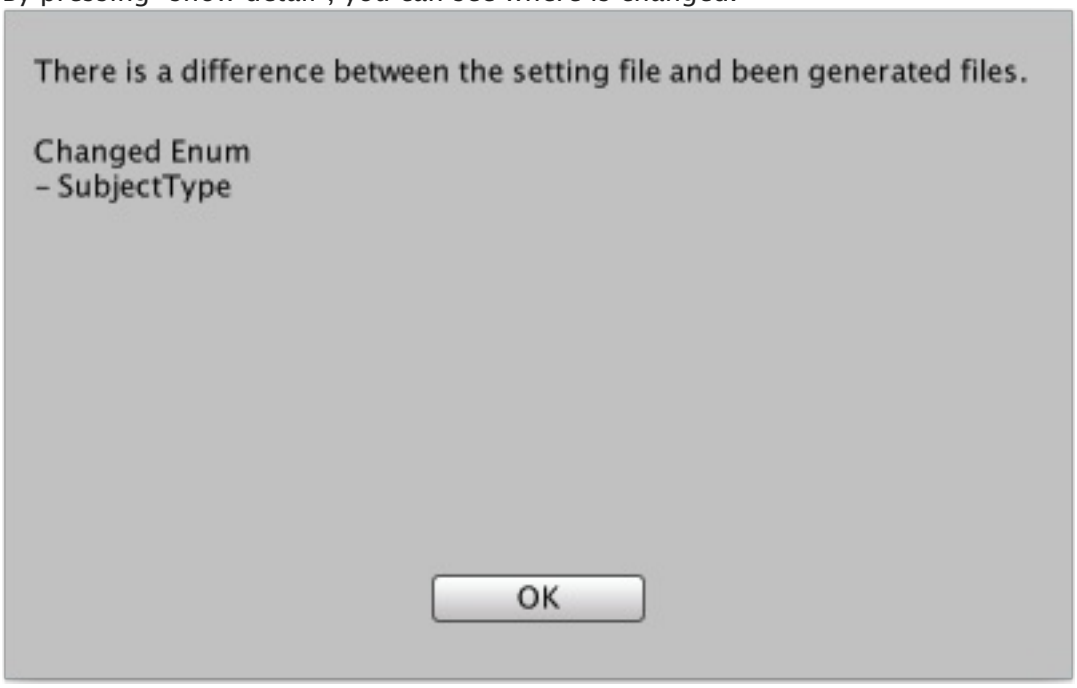
Generate script / test code

When you press the Generate button, scripts and test codes are exported.
The directory “method” contains the methods defined in the method list, “vo” contains the models defined in the model list, and “en” contains enum scripts defined in the enum list, respectively.
VoidVO in the directory “vo” is a model for when Void is specified as a method's return value.
In the directory “core” are classes for performing server connection and classes for error information.

※ WWWScriptGenerator can generate only C# and not support other langhages.

Once scripts have been generated, every time the setting is modified, or the setting file is changed with git etc, check whether there is any change from the previous setting, and if there is a change, it is noticed.

By pressing “show detail”, you can see where is changed.



Generate script

Subdirectories are created under the directory specified in Save Path of the basic setting, and the scripts are saved into them.
APITaskBase is the most important class for connecting server. it perform connecting by using "WWW" class. APITask is a class that inherits APITaskBase. Users can customize the connecting process by overriding methods in this class.

In addition, you can customize APIError and define enum for error code.

```
[core]
    APIError.cs // A class that inherits APIErrorBase
    APITask.cs // A class that inherits APITaskBase
    [basic]
        APIErrorBase.cs // For error code and message
        APITaskBase.cs // For connecting server
[en]
    <enum name>.cs
[method]
    <method name>.cs
[vo]
    <model name>.cs
```

The scripts except APIError and APITask will be overwritten every generating. On the top of those script, there is a comment like below.

```
// [Caution] This file was made by WWW Script Generator automatically. Don't
modify it.
```

As written, please do not edit these. Even if you edit it, it will be overwritten on the next generating.
However, since APIError and APITask are scripts that users can edit, they are not overwritten even if generating is performed.

After generating, if you change the namespace in basic setting, errors will occur due to mismatch between APIError and APITask namespace and other scripts. In that case, manually modify APIError and APITask, or check “Overwrite script” and re-generate to forcibly overwrite scripts. Please note that customized code will be overwritten.

Customizing APIError

This class represents an error condition, and you do not need to write logic. The type of property “Code” is integer, but it is useful to define error code as enum in APIError and cast it to int.

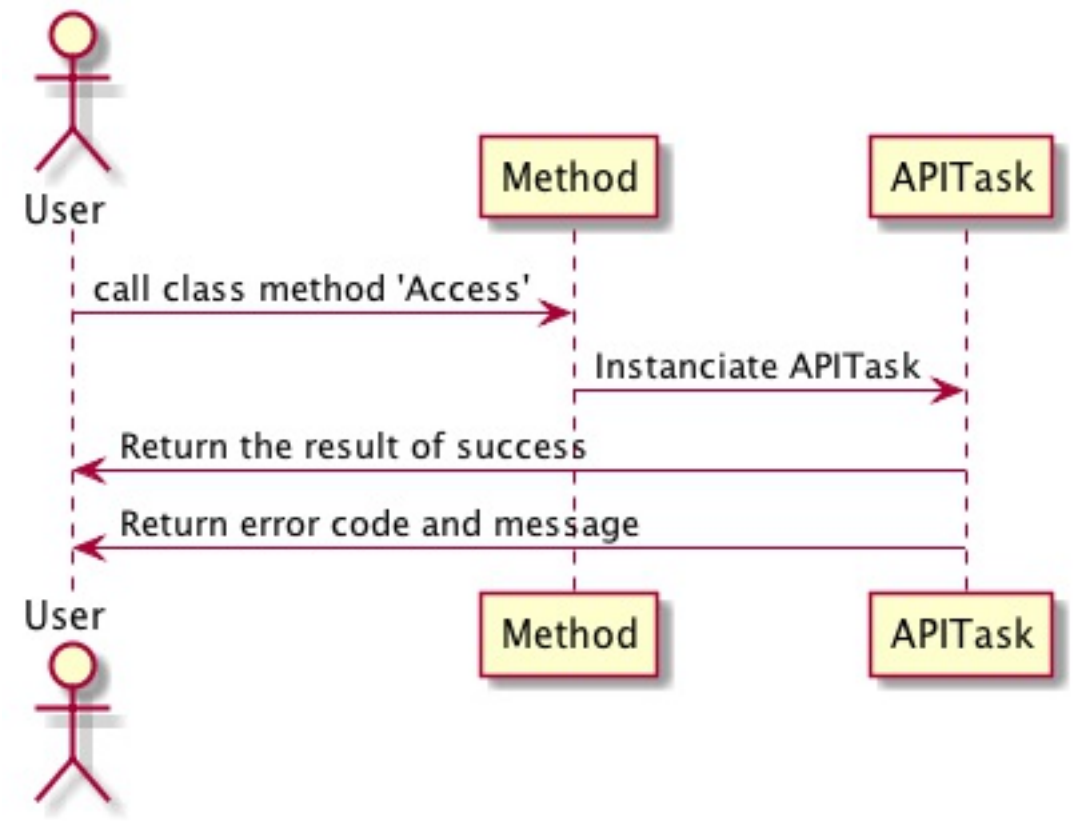
```
/// Error Data for API method
public class APIError : APIErrorBase
{
    // In most cases, when server error occurred, the server returns an error
code.
    // We define that code as an enum or static constant.
    // This is an example and you can modify it.
    //
    // public enum ErrorCodeEnum
    // {
    //     AuthorizationError = 1000,
    //     UnknownMember = 1001,
    // }
    //
    // public ErrorCodeEnum ErrorCode { get { return (ErrorCodeEnum)Code; } set
{ Code = (int)value; } }

    public APIError(int code = 0, string message = null) : base(code, message)
    {
    }
}
```

Customizing APITask

This class actually communicates with the server using the WWW class. By calling "Access" of the class method provided for each API method class, the APITask communicates and returns the result to the user. An instance of APITask is created for each communication, and it is destroyed

when the communication is finished. APITask inherits APITaskBase, and main communication logic is written in APITaskBase. User can customize the process by overriding methods and editing APITask.



- enum defined in the base class
    - APIDomainType: Domain names in domain list are defined as enum

      ```
      public enum APIDomainType
      {
          Basic,
      }
      ```

    - APIParamType: Common parameter names are defined as enum.
      public enum APIParamType
      {
      None,
      CommonParam,
      }

    - APIMethodType: Method POST/GET are defined.

      ```
      public enum APIMethodType
      {
          POST,
          GET,
      }
      ```

- Member variables of base class
  This method has Url, DomainType, ParamType, MethodType, IsList, tmpError, paramTable, paramInURLTable, paramInURLOrderList as variable, they can be used for conditional branching of processing to be customized.

- Class variables of base class

  - HandleCookie: Whether to handle cookies or not. Default values is same the one in basic setting.
  - KeyForCookie: Key for storing cookie in PlayerPrefs. Default values is same the one in basic setting.

- Class method of base class

  - ClearCookie: Delete the held cookie from Player Prefs. You can use it at the necessary timing, such as when the user logs out. e.g. 'APITask.ClearCookie();' or 'APITaskBase.ClearCookie();'

    ```
    public static void ClearCookie()
    {
    if (!string.IsNullOrEmpty(KeyForCookie) && PlayerPrefs.HasKey(KeyForCookie))
    {
    PlayerPrefs.DeleteKey(KeyForCookie);
    }

    }
    ```

- Class method of APITask

  - OverwriteSetting: Overwrite the cookie setting. Normally, you don't need to call this method because the basic setting is used as the default value. but if you want to change dynamically, use this method.

    ```
    public static void OverwriteSetting()
    {
    // You can overwrite settings such as coockies.
    // e.g.
    // HandleCookie = true;
    // KeyForCookie = "wsg_cookie";
    }
    ```

- TimeoutSec: Change timeout time. If you want to set it to 30 seconds, write as follows.

  ```
  override protected float TimeoutSec
  {
      get { return 30; }
  }
  ```

- GetCommonParamValue: Determines the value for the key of the common parameter.
  It is assumed that common parameter's values are stored in PlayerPrefs, files, etc. Use those values and return appropriate values for each key.

```
  override protected System.Object GetCommonParamValue(string key)
  {
      // If common parameter exists, you have to return value for them.
      // Values are assumed to be stored in PlayerPrefs such as member
ID.
      // e.g.
      // switch (key)
      // {
      //     case "member_id": return PlayerPrefs.GetString("member_id",
"");
      //     default: return "";
      // }
      return "";
  }
```

- AddCommonParam: Additional processing of common parameters
  In this code, temporarily hold common parameters in the table before adding it to WWWForm. In many cases, there is no need to change the processing here.

```
  override public void AddCommonParam()
  {
      // You can customize how to handle common parameters
      base.AddCommonParam();
  }
```

- AddParam: Additional processing of normal parameters
  In this code, temporarily hold parameters in the table before adding it to WWWForm. In many cases, there is no need to change the processing here.

```
  override public void AddParam(string key, System.Object value)
  {
      // You can customize how parameters are added.
      base.AddParam(key, value);
  }
```

- GetBinaryDataFileName: Specify the file name of the binary data to be added as required.

```
  override protected string GetBinaryDataFileName(string key)
  {
      // You can specify a filename for binary data.
      return base.GetBinaryDataFileName(key);
  }
```

- GetBinaryDataMimeType: Specify the Mime Type of the binary data to be added as required.

```
  override protected string GetBinaryDataMimeType(string key)
  {
      // You can specify a mime type for binary data.
      return base.GetBinaryDataMimeType(key);
  }
```

- WWWForm: Add data that was temporarily held in the table to WWWForm.
  Normally, it adds directly to WWWForm via AddField or AddBinaryData method, but if it is necessary to insert all data into one JSON, encrypt it, etc., you can change it here.

```
override protected WWWForm GetForm()
{
    // You can customize how to generate WWwForm instance.
    return base.GetForm();
}
```

- GetDomainURL: Returns the URI for the domain type.
  If the server is divided between development environment and production, you can make conditional branching here.
  The base class returns all patterns of URI defined in the domain list, you can change it to returns the appropriate URI based on the domain type of this method and the value of the preprocessor.

```
override  protected string GetDomainURL(APIDomainType domainType)
{
    // You can change the domain of url depending on conditions.
    return base.GetDomainURL(domainType);
}
```

- GetProperURL: Combines the URI of the domain and the endpoint to construct the final URL.
  If the method type is GET, the parameter is included in the URL. In many cases, there is no need to change the processing here.

```
override protected string GetProperURL()
{
    // You can change the URL depending on conditions.
    return base.GetProperURL();
}
```

- CreateHeader: If you need headers, you can create it here.
  When handring cookie, Dictionary instance for headers is created in the base class. In that case please do not create an instance here, please use the instance created in the base class.

```
override protected Dictionary<string, string> CreateHeader()
{
    // Header information can be added to form depending on conditions.
    return base.CreateHeader();
}
```

- CreateWWW: Create a WWW instance. In many cases, there is no need to change the processing here.

```
  override protected WWW CreateWWW(string properUrl, WWWForm form,
Dictionary<string, string> headerData)
  {
      // You can customize the WWW instance depending on the condition.
      return base.CreateWWW(properUrl, form, headerData);
  }
```

- CheckError: Determine if it is an error by using Look the response from the server.
  If timeout occurs or www.error is returned, error handling is done before reaching this method, but after www.isDone, this code is used to judge whether it is an error by using the return value from the server.
  For examle in one server, the value of the key “code” in JSON indicates an error code, or the value of the key “success” is false indicates an error occured. It depends on the implementation of the server, but in case of error, an error code and message will be returned in many cases. If an error occurs, create an instance of APIError, set the properties of “Code” and “Message”, and return that instance. If it succeeds it returns null.

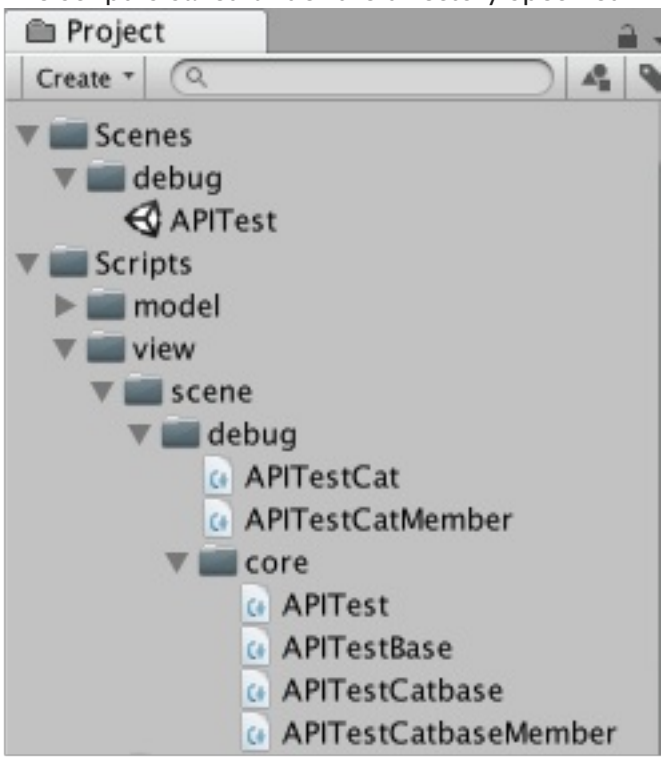
```
  override protected APIError CheckError(WWW www)
  {
      // You can change error codes and messages depending on the server
side specifications.
      // In this example, assuming that "result" and "message" will be
returned from the server.
      // If the value of "result" is 0, it means succeed, otherwise
failure.
      //
      // var jsonStr = Regex.Unescape(www.text);
      // var vo = JsonUtility.FromJson<CommonVO>(jsonStr);
      // if (vo.Result == 0) return null;
      // var err = new APIError(vo.Result, vo.Message);
      // err.ErrorType = APIErrorType.Other;
      // return err;
      return base.CheckError(www);
  }
```

- AdjustReturnValue: Here you can adjust the return value when communication is successful.
  In many cases, there is no need to change the processing here.
  “retVal” has already parsed www.text to the appropriate return class, but you can adjust here if the return value is binary or if there is additional information.

```
  override protected void AdjustReturnValue(System.Object retVal, WWW
www)
  {
      // retVal is VO class which is the return value of method parsed
with www.text.
      // You can adjust the return value by using www before it passed to
the success handler.
  }
```

Generate test code

Scene file is saved under the name specified in “Test code name” under the directory whose name specified in “Test code scene path”.
The script is saved under the directory specified in “Test code script Path”.



A category of methods is the left-hand string if Endpoint is delimited by “/”.

```
APITestCat.cs // A class that inherit APITestCatbase
APITestCat<category of methods>.cs // Classes that inherit
APITestCatbase<category of methods>
[core]
    APITest.cs // main script attached to the scene file
    APITestBase.cs // base class of APITestCatbase.cs and APITestCatbase<method
category>.cs which common processing is written
    APITestCatbase.cs // Test code of method whose Endpoint is not delimited by
“/”
    APITestCatbse<category of methods>.cs // test code of the method whose
Endpoint is separated by “/”
```

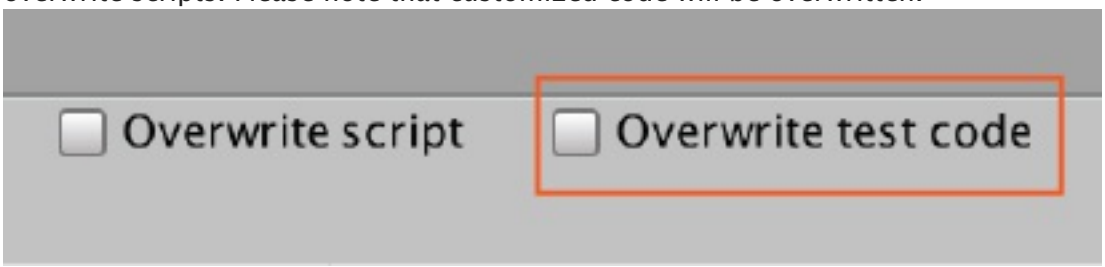
The scripts except `APITestCat.cs` and `APITestCat<category of methods>.cs` will be overwritten every generating. On the top of those script, there is a comment like below.

```
// [Caution] This file was made by WWW Script Generator automatically. Don't
modify it.
```

As written, please do not edit these. Even if you edit it, it will be overwritten on the next generating.
However, since `APITestCat.cs` and `APITestCat<category of methods>.cs` are scripts that users can edit, they are not overwritten even if generating is performed.

After generating, if you change the namespace in basic setting, errors will occur due to mismatch between `APITestCat` and `APITestCat<category of methods>` namespace and other scripts. In that case, manually modify them, or check “Overwrite test code” and re-generate to forcibly

overwrite scripts. Please note that customized code will be overwritten.



Customize APITestCat.cs, APITestCat＜category of methods＞.cs

For one server method, two methods are created: Test<server method name>Execute and Test<server method name>Success. For example, in the case of a method named ”Scorelist”, TestScorelistExecute and TestScorelistSuccess are created.

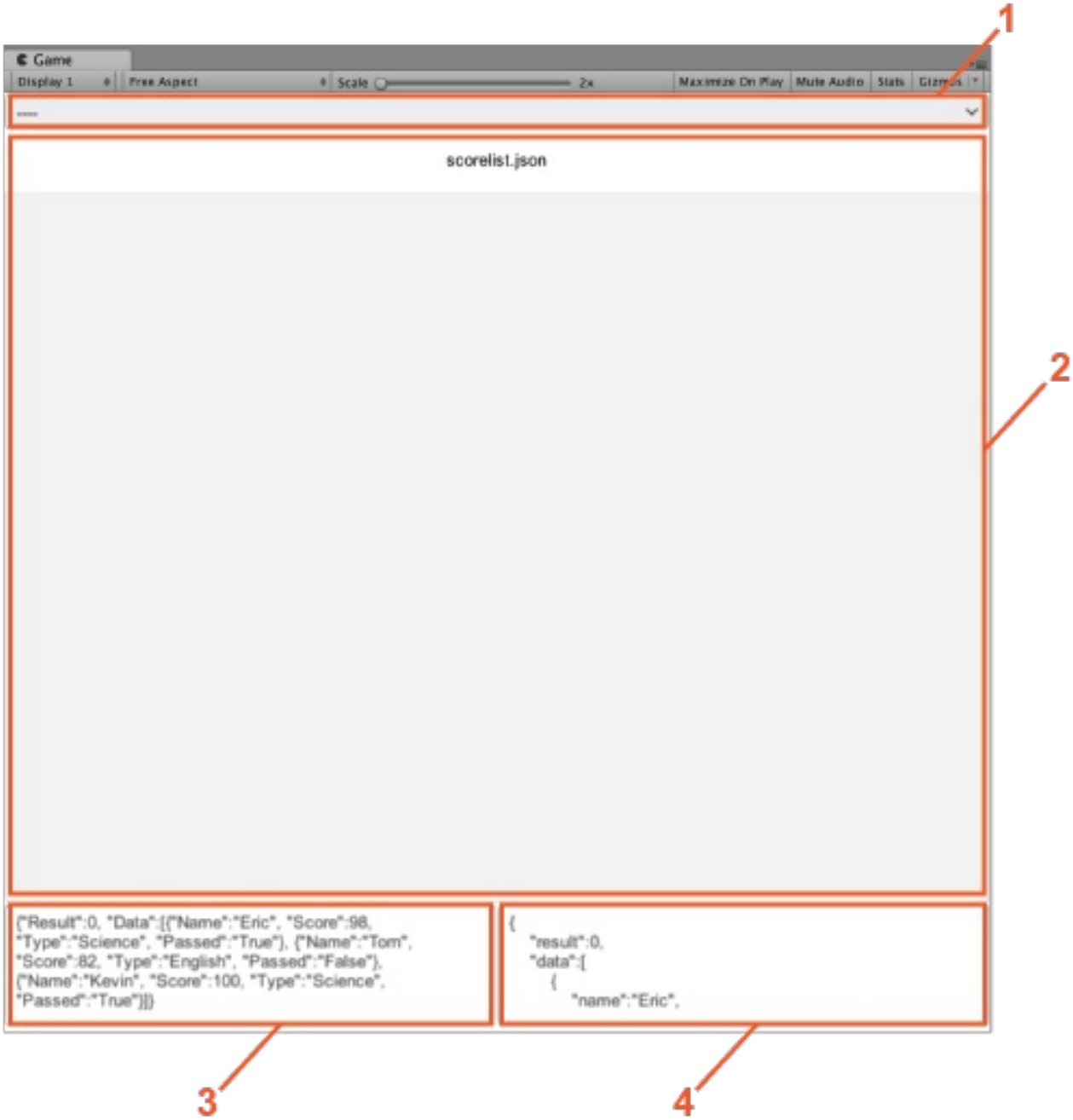
- Test<server method name>Execute: It is called when test execution.
  Since the value of the parameter is not specified in the code just generated, specify an appropriate value.
  override protected void TestScorelistExecute(string memberId)
  {
  // Set these parameters for method.
  memberId = ""; // Member's ID
  base.TestScorelistExecute(memberId);
  }

- Test<server method name>Success: It is called on success.
  If you want to save the return value in PlayerPrefs, you can write it here.

```
  override protected void TestScorelistSuccess(ScoreListVO vo)
  {
      // If you want to save the property of the return value, you can
write it here.
  }
```

Execution of test code

You can run the test by playing on Unity Eitor. Also, if you build it you can check it on the actual devices.

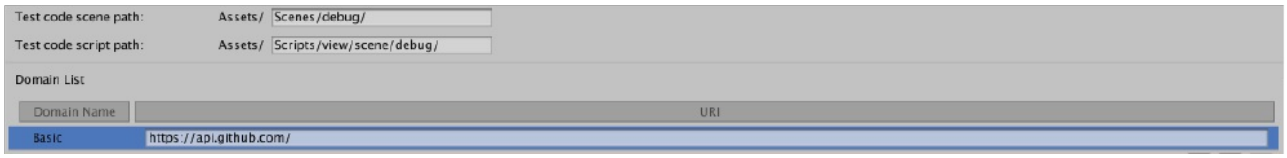


1. Select a category of methods. A category of methods is the left-hand string if Endpoint is delimited by "/". If it is not delimited by "/" it is classified as "----".

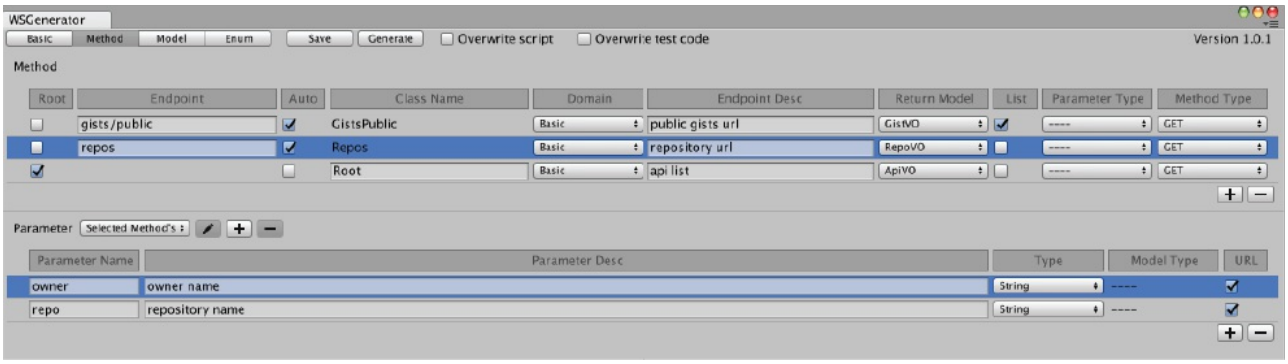
2. Display list of methods. Each item is a button, and by pressing it executes the method.
3. Display the results of the execution. If there is a return value, it displays the content of that model.
4. The character string actually returned from the server is displayed as it is. This will be a clue to investigate the cause if it did not result as expected.

Try the tutorial data

Let's actually try using the tutorial data.
In the basic setting screen, press the Create Tutorial Data button to create tutorial data.
In the Domain List, make sure that the URI of the domain named Basic is `https://api.github.com/`. In this tutorial we use github's API.



When you move to the method screen by pressing the Method tab, you will find three methods are created.

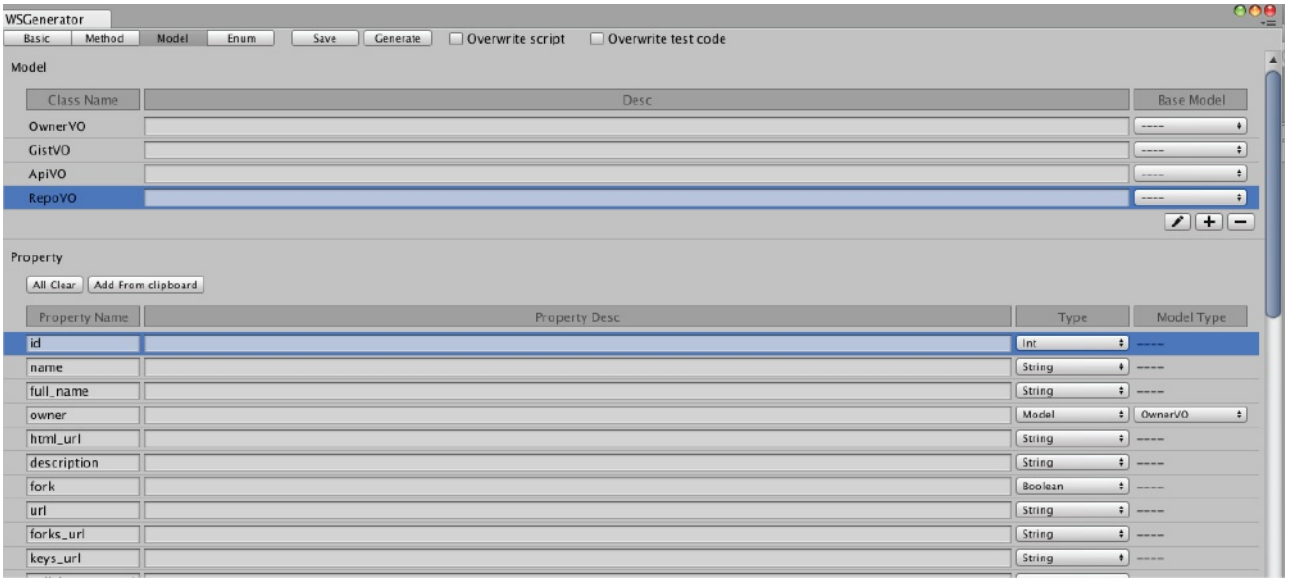


Let's look at the line whose Endpoint is gists/public. It's Class Name is GistPublic. This is the name automatically created from Endpoint because Auto is checked.
Return Model's value "GistVO" indicates that the model parsing JSON of the return value of this method is GistVO.
List is checked and it means that direct array is returned, and the JSON of the return value starts with [.

Next, click on the left edge of the line whose Endpoint is "repos". The line is selected and it becomes blue, and two parameters, owner and repo, appear. This means that this method need two parameters to be called. And the URL of each parameter is checked. This means that the parameters are directly inserted in the URL, delimited by "/".

There is no Endpoint in the method below it. Instead, Root is checked. This means this method calls the domain `https://api.github.com/` as it is.

When you move to the model list screen by pushing the Model tab, you will find four models are created.

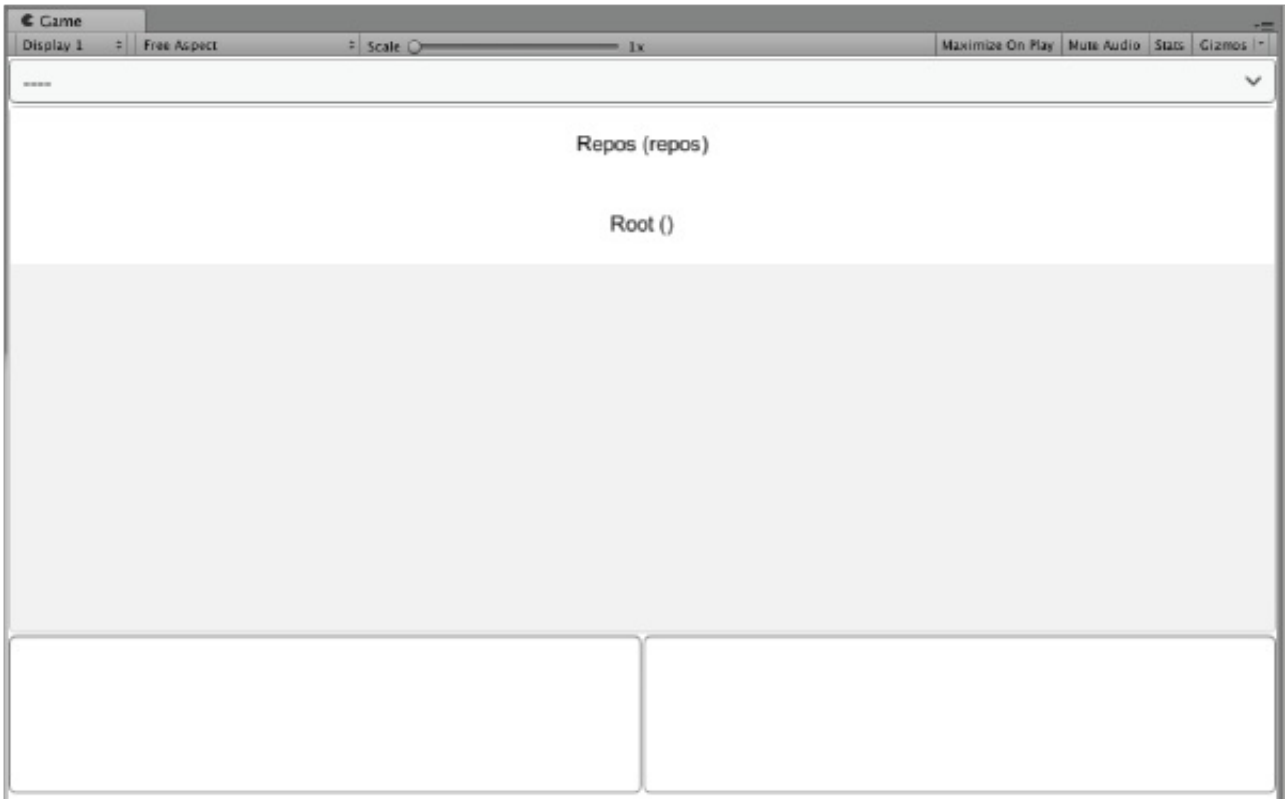


Looking at the property "owner" of GistVO, the Type is "Model" and Model Type is "OwnerVO". This means that OwnerVO is refered from GistVO. Likewise, in RepoVO, the type of the property "owner" is "OwnerVO", and it means this property refers "OwnerVO".

Make sure Generate Test Code is checked on the basic setting screen, and press the Generate button on the common menu. Scripts and test code are written out. Open the scene file "APITest" in Assets/Scenes/debug and play it with the play button of the unity editor.

On screen, belows are shown.

- Repos (repos)
- Root ()



Click on "Root" and the result will be displayed at the bottom after a while.

{"CurrentUserUrl":"https://api.github.com/user",
"CurrentUserAuthorizationsHtmlUrl":"https://github.com/settings/connections/applications{/client_id}",
"AuthorizationsUrl":"https://api.github.com/authorizations",
"CodeSearchUrl":"https://api.github.com/search/code?q={query}

{
"current_user_url": "https://api.github.com/user",
"current_user_authorizations_html_url":
"https://github.com/settings/connections/applications{/client_id}",
"authorizations_url": "https://api.github.com/authorizations",

"Repos" will fail even if it is pressed as it is. It is because the parameters are not set. To set the parameters you need to customize the test codes, and these are written for each of the categories which discribed above. Since the endpoints of these two methods are not delimited by "/", there is no category name. Therefore, because the test code name does not have a suffix, "APITestCat.cs" is the name of the target code.

```
override protected void TestReposExecute(string owner, string repo)
{
    // Set these parameters for method.
    owner = ""; // owner name
    repo = ""; // repository name
    base.TestReposExecute(owner, repo);
}
```

Since "repo" and "owner" in this method are blank, specify owner name of github and repository name. (Ex: owner = "inosyan" repo = "ScrachBand2")
And play again, the result will appear.

In the pull-down menu at the top, if you switch the category from ---- to gist, GistsPublic (gists / public) will be displayed.

When you execute, An error "String too long for TextMeshGenerator. Cutting off characters." is appear. Please do not worry it. because it is from uGUI problem, it means there are many characters to display. The return value is displayed with no problem.

When you open APITestCatGists.cs,

```
override protected void TestGistsPublicSuccess(GistVOList vo)
```

the argument type is GistVOList. This is a class not defined in the model, but since the return value of this method is array of "GistVO" which you specified by checking "List" on the method screen, it is a class automatically created to store the return value. The JSON parse result is stored in the property "List".

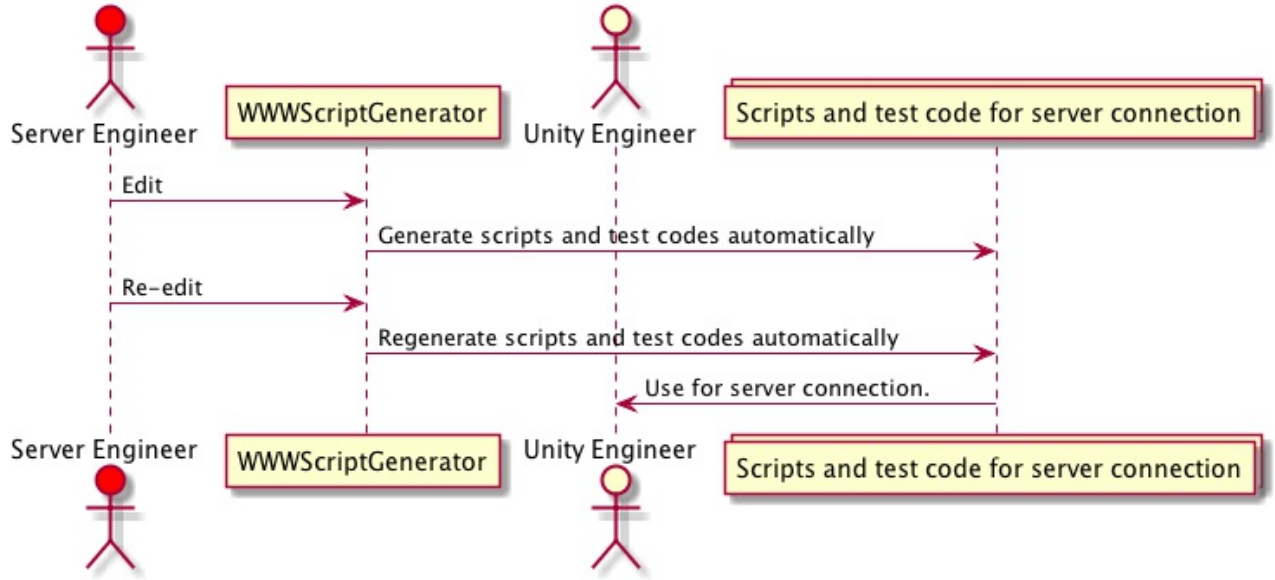
Let's look at the base class APITestCatbaseGists.cs. Methods for API are using in the codes like "TestGistsPublicExecute()". Please refer to when actually using the method. The type of vo is the Return Model set on the method screen.

```
ApiGistsPublic.Access((vo, rawText) =>
    {
        // Success processing
    },
    (err, rawText) =>
    {
        // Error processing
    }, gameObject);
```

How to share settings with server engineer

Sharing setting files

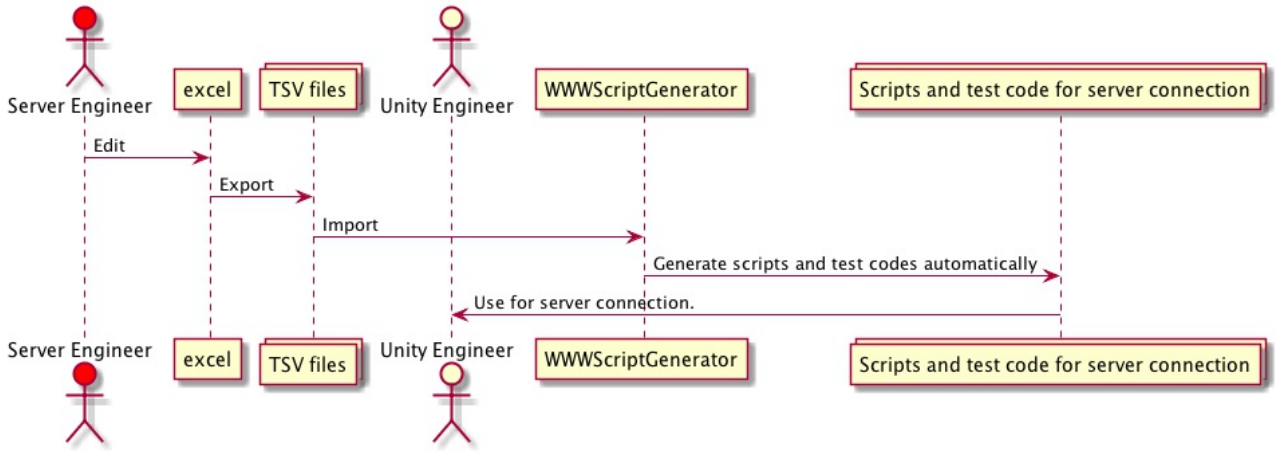
By sharing the setting file, it is possible to edit settings mutually.
“wwwscriptgenerator_setting.json” in “Assets/WWWScriptGenerator/UserData/” is the setting file.
Share this file with git etc. and edit it with Unity.



Sharing Excel file (Pro version only)

It is possible to edit settings mutually through Excel file.
“wsg_template.xlsm” in “Assets/WWWScriptGenerator/” is the Excel file. This file is original and you should keep it and make copy name such as wsg.xlsm in directory “UserData”. Share the file with git etc, export the TSV from Excel, and load it to Unity.

This Excel file has input validation with macro, problems due to typing errors can be prevented beforehand.

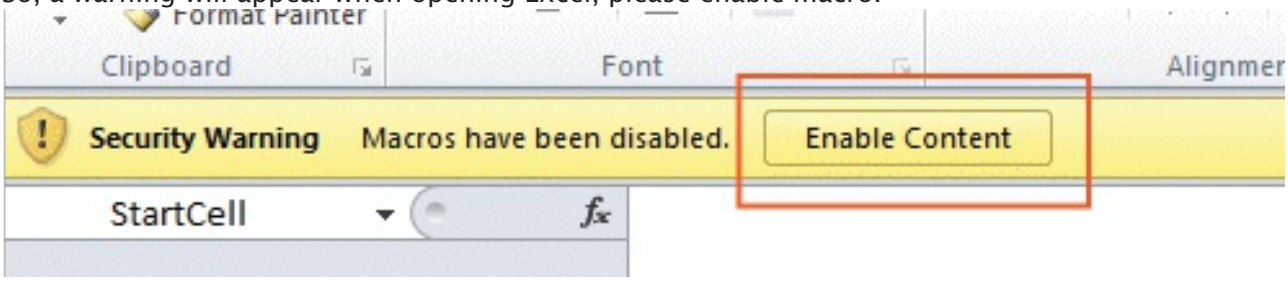


Sharing TSV file (Pro version only)

By sharing the TSV file directly, it is possible to edit settings mutually. However, there is no input validation as in the case of Excel, you need to be careful about input errors.

About Excel file

Excel includes macro for TSV import / export and input validation.
So, a warning will appear when opening Excel, please enable macro.

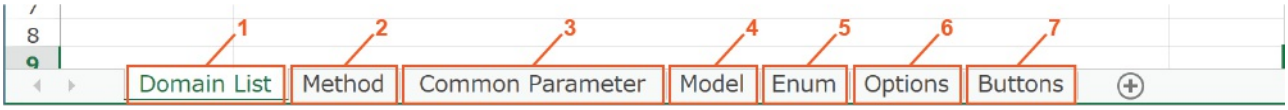


We assume this Excel file will be shared between server engineers and Unity engineers.

1. Server engineer edits Excel file, Unity engineer receives it and imports it to Unity
2. If you have a prototype project, Unity engineer exports settings from Unity and passes Excel to the server engineer
   In either case, it is possible to mutually edit between Excel and Unity.

Sheets

In Excel it is divided into sheets. Only Unity settings that need to be shared with server engineers are included.

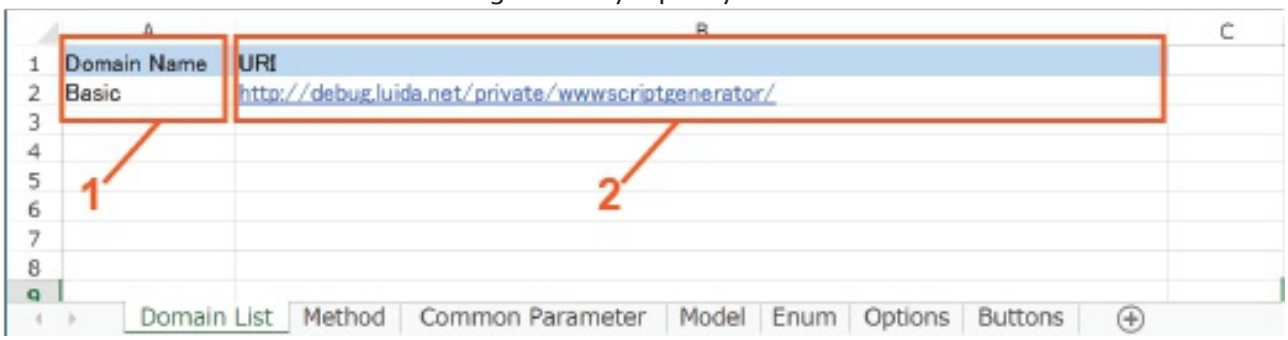


1. Domain List: This is the Domain List of basic settings in Unity. Specify the server connection destination.
2. Method: Method list in Unity.
3. Common Parameter: Common parameter of method list in Unity.

4. Model: Model list in Unity.
5. Enum: enum list in Unity.
6. Options: Sheet used for input validation of Excel. (User does not need to edit)
7. Buttons: This sheet includes buttons for executing macro.

Domain list

This is the Domain List of basic settings in Unity. Specify the server connection destination.



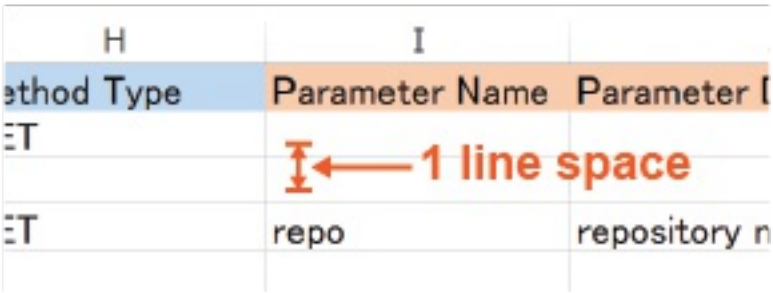
For one Domain Name, specify one URI. Write items without spacing lines.

1. Domain Name: Name of the domain
2. URI: URI of domain

Method

Unity method list.
Auto and Class Name in Unity's method list are items used only on the Unity side, so they are not included in Excel.
! [] (images / ai_export / excel_method.png)
For each Endpoint, specify Domain, Endpoint Desc, Return Model, Parameter Type, and Method Type one by one.
Multiple parameters can be specified for one Endpoint. For each Parameter Name, specify Parameter Desc, Type and Model Type one by one.

You can insert one line space between Parameter Name and Parameter Name of the following method for easy viewing. There is no problem even if there is no space.



1. Root: Check this if you do not have an endpoint and you want to use the URI specified in the Domain list as it is.

2. Endpoint: The endpoint of the method. Write a string following the URI specified in the Domain list.

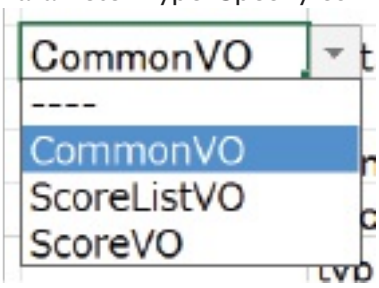
3. Domain: Specify the domain URI to use from the domain list.

4. Endpoint Desc: It is for describing what purpose this endpoint is, it will be a comment in the generated script.

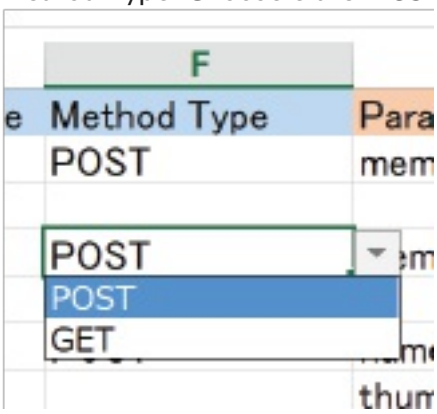
5. Return Model: Specify the return value of method from models. If there is no return value, select “Void”.

6. List: It is specified when the return value of JSON is an array.
   For example, if JSON is
   `[{"name":"a"}, {"name":"b"}]`
   instead of
   `{"list":[{"name":"a"}, {"name":"b"}]}`
   , set "TRUE".

7. Parameter Type: Specify common parameters. If you do not specify it, choose “----”.



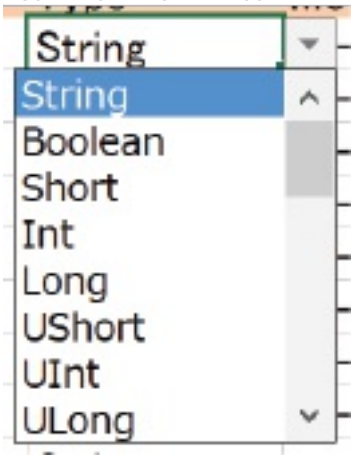
8. Method Type: Choose either POST or GET for connection method.



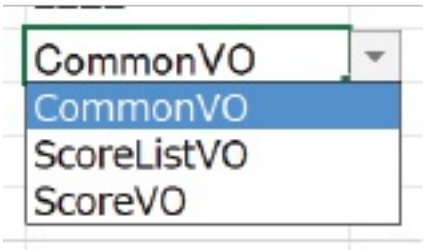
9. Parameter Name: Specify the parameter name. It is the same as the key name sent to the server when actually communicating with the server.

10. Parameter Desc: It is for describing the parameter, it will be a comment in the generated script.

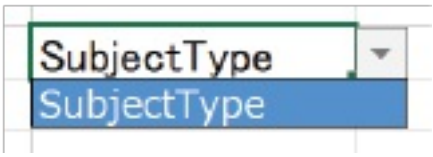
11. Type: Specify the type of the parameter.



12. Model Type: Choose which type to use when the type is `Model`, `Enum`, `List<Model>`, `List<Enum>`.
    For `Model`, `List<Model>`, choose from Model.



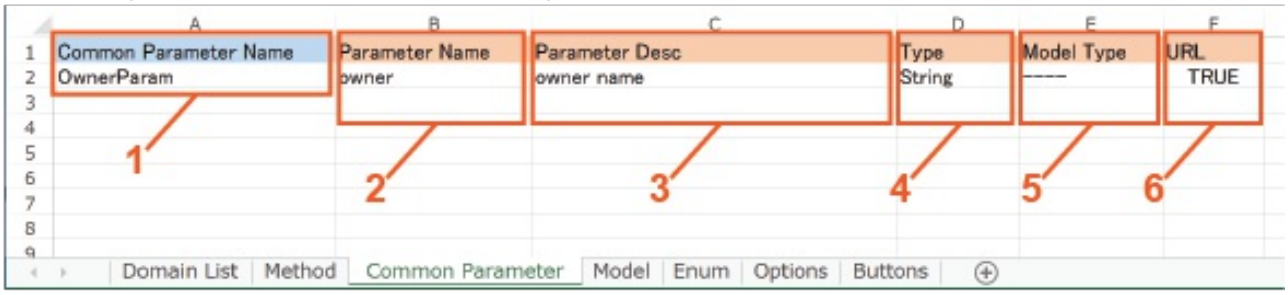
    For `Enum`, `List<Enum>`, choose from Enum.



13. URL: Whether the parameter is in the URL
    For example, if the value of the parameter "param1" is "value1", the actual URL is `https://example.com/value1`.

Common parameter

Common parameter of method list in Unity.

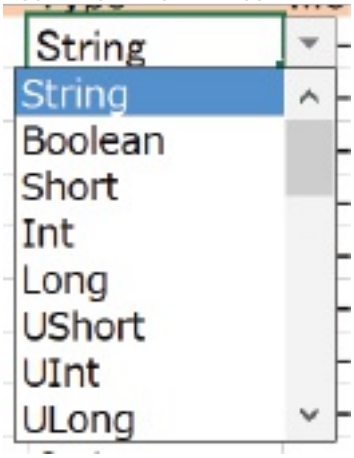


Multiple parameters can be specified for one Common Parameter Name. For each Parameter Name, specify Parameter Desc, Type and Model Type one by one.

You can insert one line space between Parameter Name and Parameter Name of the following common parameter for easy viewing. There is no problem even if there is no space.

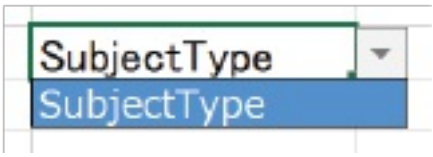
1. Common Parameter Name: Specify the name of common parameter

2. Parameter Name: Specify the parameter name. It is the same as the key name sent to the server when actually communicating with the server.
3. Parameter Desc: It is for describing the parameter, it will be a comment in the generated script.
4. Type: Specify the type of the parameter.



5. Model Type: Choose which type to use when the type is `Model`, `Enum`, `List<Model>`, `List<Enum>`.
   For `Model`, `List<Model>`, choose from Model.

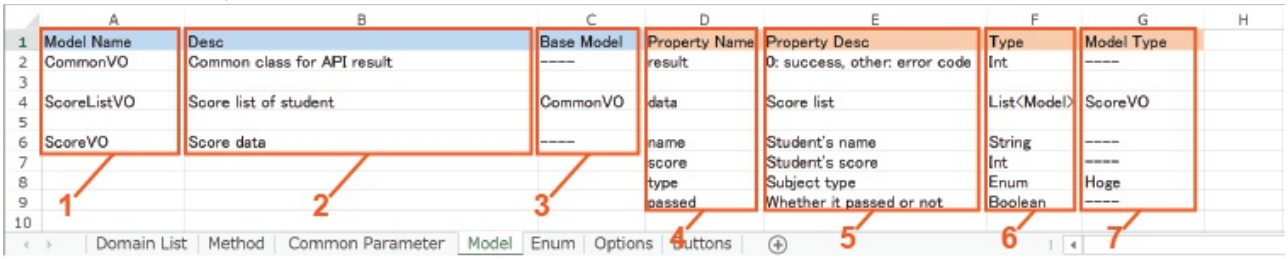
   For `Enum`, `List<Enum>`, choose from Enum.



6. URL: Whether the parameter is in the URL
   For example, if the value of the parameter "param1" is "value1", the actual URL is `https://example.com/value1`.

Model list

Model list in Unity.

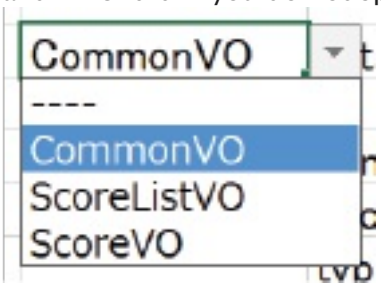


Model Name	Desc	Base Model	Property Name	Property Desc	Type	Model Type
CommonVO	Common class for API result	----	result	0: success, other: error code	Int	----
ScoreListVO	Score list of student	CommonVO	data	Score list	List<Model>	ScoreVO
ScoreVO	Score data	----	name	Student's name	String	----
			score	Student's score	Int	----
			type	Subject type	Enum	Hoge
			passed	Whether it passed or not	Boolean	----

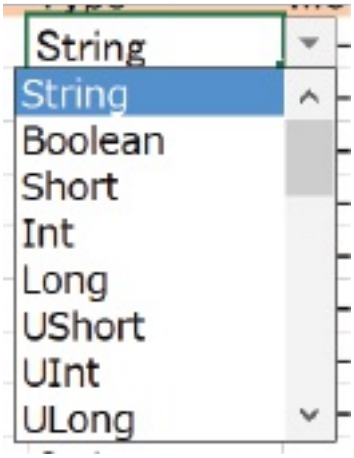
For each Model Name, specify Desc and Base Model one by one.
Multiple property can be specified for one Model Name. For each Property Name, specify Property Desc, Type and Model Type one by one.

You can insert one line space between Property Name and Property Name of the following model for easy viewing. There is no problem even if there is no space.

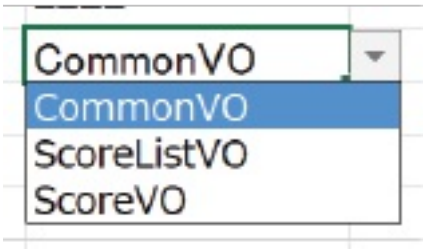
1. Model Name: Specify the class name of the model.
2. Desc: It is for descripting the model, it will be a comment in the generated script.
3. Base Model: If there is a model with common properties, you can create a common class and inherit it. If you do not specify it, choose "----".



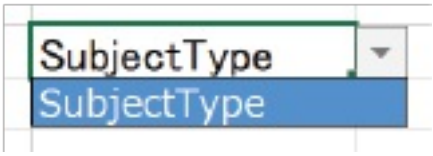
4. Property Name: Specify the property name. Make it the same as the JSON key name when actually communicating with the server.
5. Property Desc: It is for describing the property, it will be a comment in the generated script.
6. Type: Specify the type of the parameter.



7. Model Type: Choose which type to use when the type is `Model`, `Enum`, `List<Model>`, `List<Enum>`.
   For `Model`, `List<Model>`, choose from Model.

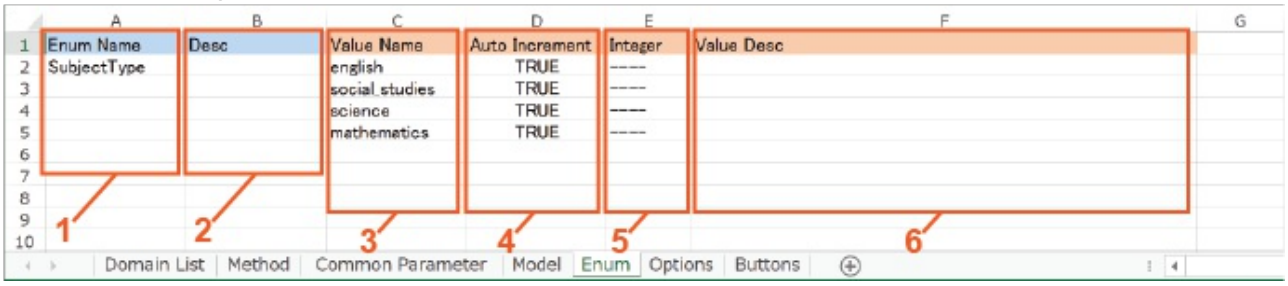


   For `Enum`, `List<Enum>`, choose from Enum.



enum list

enum list in Unity.



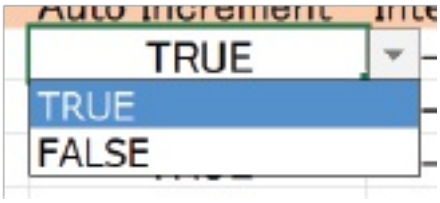
	A	B	C	D	E	F
1	Enum Name	Desc	Value Name	Auto Increment	Integer	Value Desc
2	SubjectType		english	TRUE	----	
3			social_studies	TRUE	----	
4			science	TRUE	----	
5			mathematics	TRUE	----	

Domain List | Method | Common Parameter | Model | Enum | Options | Buttons

For each num Name, specify Desc one by one.
Multiple enum values can be specified for one Enum Name. For each Value Name, specify Auto Increment Integer and Value Desc one by one.

You can insert one line space between Value Name and Value Name of the following enum for easy viewing. There is no problem even if there is no space.

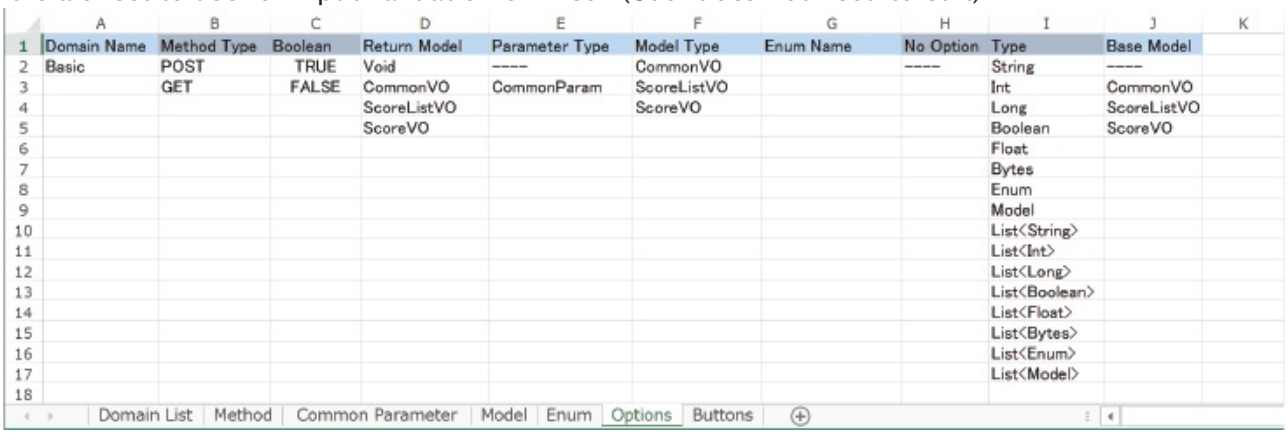
1. Enum Name: Specify the name of enum
2. Desc: It is for descripting the enum, it will be a comment in the generated script.
3. Value Name: Specify the name of the enum's value. Since it is converted to Pascal in the exported code, it does not have to be capitalized here.
4. Auto Increment: Set False if you want to specify an integer value. If True, the value of the integer is the previous value plus one.
5. Integer: If you want to specify an integer value, set Auto Increment to False and specify a value.



6. Value Desc: It is for describing the enum's value, it will be a comment in the generated script.

Options

It is a sheet to use for input validation of Excel. (User does not need to edit)



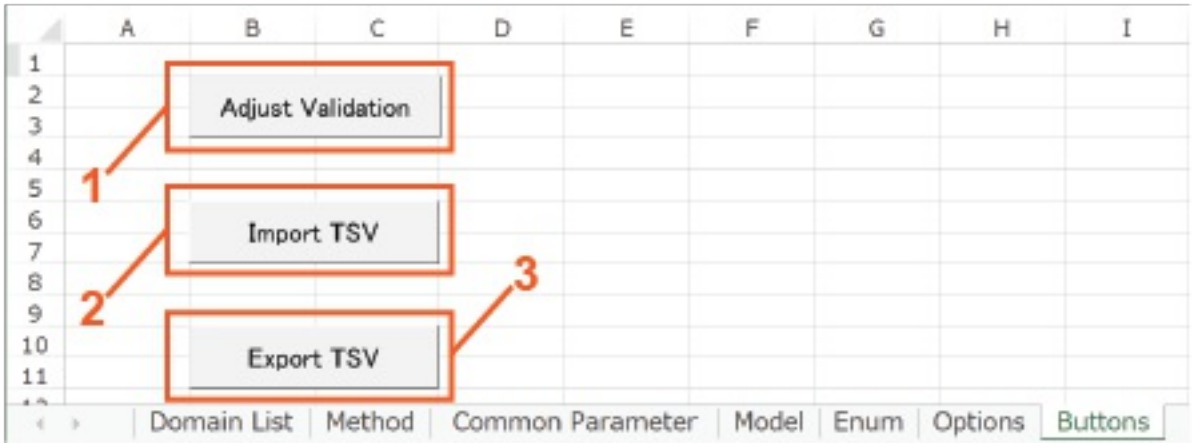
	A	B	C	D	E	F	G	H	I	J
1	Domain Name	Method Type	Boolean	Return Model	Parameter Type	Model Type	Enum Name	No Option	Type	Base Model
2	Basic	POST	TRUE	Void	----	CommonVO		----	String	----
3		GET	FALSE	CommonVO	CommonParam	ScoreListVO			Int	CommonVO
4				ScoreListVO		ScoreVO			Long	ScoreListVO
5				ScoreVO					Boolean	ScoreVO
6									Float	
7									Bytes	
8									Enum	
9									Model	
10									List<String>	
11									List<Int>	
12									List<Long>	
13									List<Boolean>	
14									List<Float>	
15									List<Bytes>	
16									List<Enum>	
17									List<Model>	

Domain List | Method | Common Parameter | Model | Enum | Options | Buttons

Macro execution buttons

It is a sheet includes buttons for executing macro.
Each TSV file is equivalent to the contents of the following sheet.

- wsg_domain.tsv: Doman List
- wsg_common.tsv: Common Parameter
- wsg_method.tsv: Method
- wsg_model.tsv: Model
- wsg_enum.tsv: Enum



1. Adjust Validation: Apply input validation to each sheet. If input validation disappears when deleting lines or copying, you can reapply it by executing this macro.
2. Import TSV: Import TSV files.
3. Export TSV: Export the TSV files.

Other

Customization of input validation

In the place where the user inputs characters, input validation is provided avoiding invalid characters included. You can customize it.
Please redefine the regular expression for input validation in the method of PrepareGUICommonResources in WSGMainWindow.cs.

- For basic setting
    - NamespaceRegex: Namespace input
    - PrefixRegex: Enter prefix
    - SavePathRegex: Enter save path
    - TestCodeNameRegex: Enter test code name
    - TestCodeNameSpaceRegex: Enter test code namespace
    - TestCodeScenePathRegex: Enter the save path of the scene of the test code
    - TestCodeScriptPathRegex: Enter test code save script path
    - DomainNameRegex: Enter domain name
    - DomainURIRegex: Enter URI of domain
- For method list
    - EndpointRegex: Endpoint input
    - ParamNameRegex: Enter parameter name
    - CommonParameterNameRegex: Enter common parameter name
- For model list

    - ModelNameRegex: Enter model name
    - PropertyNameRegex: Enter property name

- For enum list

    - EnumNameRegex: Enter enum name
    - EnumValueRegex: Enter the value name of the enum