PLUGIN REFERENCE GUIDE

VERSION 1.02

September 10, 2023

This document is still in progress.

This document is intended to assist in the creation of plugins as used in **GURPS**® Character Assistant 5.

CONTENTS

What a Plugin Is 4
The Printer Document Object 4
Print/Export Sheet Plugin Processing Within GCA 5
Compilable Plugins 6
Compile.XML 7
AssemblyInfo.vb 8
OfficialCharacterSheet.vb 9
Required Interfaces 11
IBasicPlugin 12
Properties 12
Procedures 12
ICalledPlugin 14
Inherits 14
Events 14
Procedures 15
IExportSheet 17
Inherits 17
Events 17
Procedures 17
IPrinterSheet 20
Inherits 20
Events 20
Procedures 20
IUnifiedViewBox 22
Inherits 22
Events 22
Properties 24
Procedures 24
Objects 27
ArmorLayer 27
Body 28
BodyItem 29
DialogOptions_RequestedOptions 33
Properties 33
LayerItem 34
LoadOut 35
Party 38
Properties 38
Helpful Objects 39
FileWriter 40
Enumerations 40
Properties 40
Procedures 40
FootnoteManager 42
Enumerations 42
Properties 43
Procedures 43

GroupedTraitListBuilder 46
Properties 46
Procedures 47
ProtectionPaperDollSettings 49
TraitListGroupingOptions 51
Enumerations 51
Properties 51
Procedures 52
TraitTypes Enum 54
Disorganized notes for Future Sections and Content 55
Things Writers of Plugins Should Know 55
Do you have any documentation on the GCACharacter.Items list? 56
How do I access Loadouts? 57
How do I show the graphic with the various body locations and armor? 58
How do I access the body type, body plan, body parts, or Hit Table for the character? 58

What a Plugin Is

Plugins in GCA are DLLs written in a supported .Net language. You can compile them yourself, or you can leave them as source code and GCA can compile them for use.

You can get a free copy of Visual Studio Community, which is an integrated development environment for .Net, here:

https://visualstudio.microsoft.com/vs/community/

If you don't need the fancy features of a full-on environment, there are a number of code-editors out there that provide helpful features, such as syntax highlighting/coloring. One that I rather like for quick edit jobs is NotePad++, which includes support for both the C# and Visual Basic languages. You can get it here:

https://notepad-plus-plus.org/

I write all my plugins in Visual Basic, but I know other folks that use C#. I don't think any other languages currently have automatic support in the .Net systems available on Windows.

If you have experience with .Net and Visual Studio, you can compile your plugins to DLLs and make them available that way. If you do that, you just need to provide the DLL and have users place it into a folder under their \Documents\GURPS Character Assistant 5\plugins\ folder. GCA will pick it up on loading.

Alternatively, you can provide plugins as source code, and GCA will compile them upon loading. If you do that, you need to provide at least three different files, two specifically for the compilation process. Users will also need to place these files into a folder under their \Documents\GURPS Character Assistant 5\plugins\ folder. More details on compileable plugins is available later in the document. Note that leaving your plugins as source code has the distinct advantage of not needing you to touch them, and your users to redownload them, every time GCA's external-facing object model or third-party components are updated.

This isn't the place to teach you how to program using .Net and Visual Studio, I'm afraid. If you're familiar with scripting, and you want to create a plugin, chances are that you can look at an existing source-code plugin and figure out how to modify things to get what you are after.

There is no convenient method of writing and testing plugins without loading GCA and seeing what happens. I am sorry about that, as that can be an excruciating to way build sheets.

The Printer Document Object

Printer Sheet plugins work on a document object and that document may be sent to a printer or to a window as a preview. Export Sheet plugins have no such basic object, and just output to a file. Because of the extra work required to handle Printer Sheets, GCA has some outside help.

Reports for WinForms is a third-party tool licensed from ComponentOne. GCA uses this tool for the print and previewing features, and its object model is central to the printing process of Printer Sheet plugins.

Of specific importance is the C1PrintDocument component, since that's the document that your plugin will be constructing to send to the printer or to the preview window.

You can probably learn enough to do what you need to by studying or modifying other printer sheets, but documentation from ComponentOne can be found online. Here is a good place to start:
https://www.grapecity.com/componentone/docs/win/online-report/workingwithc1printdo.html

Print/Export Sheet Plugin Processing Within GCA

When GCA starts up, it loads all the plugins that are not specifically being excluded.

When the user asks GCA to run a plugin, such as when you switch to a new character sheet or when you choose to export, it follows the following procedure.

1) If an internal SheetOptionsManager doesn't yet exist for some reason, GCA creates one.
2) When the SheetOptionsManager is created, GCA sets the PluginHomeFolder property on the SheetOptionsManager to the folder where the sheet lives.
3) The UpgradeOptions procedure is called.
4) The CreateOptions procedure is called. This initializes all options to default values.
5) Options are then set to values found in the saved Sheet Options preferences for that sheet profile, if any.
6) If the sheet is running in Sheet View, the InSheetView property of the SheetOptionsManager is set to `True`, otherwise it is `False`.
7) A handler is created for RequestRunSpecificOptions.
8) The sheet's PreviewOptions function is called, and the sheet is aborted if True was returned.
9) If it's an export sheet, the Save dialog is shown to the user.
10) The sheet's Generate function is run.

COMPILABLE PLUGINS

GCA can take source code files and compile an assembly using the .Net compiler. It will then use that assembly to create an instance of the related plugin. This allows for providing plugins as source files which can then be compiled on the user's machine during program startup. This helps to avoid versioning problems related to upgrading assemblies within GCA.

The compiled DLL is written to disk in the appdata\plugins folder. This is considered a temporary file, and a new version is created every time GCA starts up.

COMPILE.XML

This file tells GCA what information it needs to provide to the .Net compiler so that it can do its job properly.

```
<?xml version="1.0" encoding="utf-8"?>
<plugin xmlns="gca5plugin">

    <author>
    </author>

    <language>VB</language>
    <output>OfficialCharacterSheet.dll</output>

    <source>OfficialCharacterSheet.vb</source>
    <source>AssemblyInfo.vb</source>

    <reference>System.dll</reference>
    <reference>System.Core.dll</reference>
    <reference>System.Drawing.dll</reference>
    <reference>System.Xml.dll</reference>
    <reference>System.Xml.Linq.dll</reference>
    <reference>System.Collections.dll</reference>

    <reference>Microsoft.VisualBasic.dll</reference>

    <reference>%app%\C1.C1Report.4.dll</reference>
    <reference>%app%\GCA5Engine.dll</reference>
    <reference>%app%\GCA5.Interfaces.dll</reference>

</plugin>
```

You need to include all the references that are needed, including many that are usually handled for you automatically by Visual Studio. Here's an example from the OfficialCharacterSheet plugin.

The `<language>` block tells us what language the source files are in.

The `<output>` block tells us the name of the output DLL that will be created.

The `<source>` blocks are the necessary source code files that will be included in the assembly. You should always have at least one source code file, and a file that includes information for the assembly that the compiler will need.

The `<reference>` blocks include needed assembly references for the project. The last three shown here are the ones needed specifically from within GCA for a print sheet plugin.

AssemblyInfo.vb

This is the file that includes information detailing properties of the assembly, which is the DLL containing the plugin. In this case, this is the file name used by Visual Basic; your language may use something different, but so long as you include a `<source>` block for it in Compile.XML it should be fine.

```
Imports System
Imports System.Reflection
Imports System.Runtime.InteropServices

' General Information about an assembly is controlled through the following
' set of attributes. Change these attribute values to modify the information
' associated with an assembly.

' Review the values of the assembly attributes

<Assembly: AssemblyTitle("OfficialCharacterSheet")>
<Assembly: AssemblyDescription("")>
<Assembly: AssemblyCompany("")>
<Assembly: AssemblyProduct("OfficialCharacterSheet")>
<Assembly: AssemblyCopyright("Copyright © 2016-2020 Armin D. Sykes")>
<Assembly: AssemblyTrademark("")>

<Assembly: ComVisible(False)>

' Version information for an assembly consists of the following four values:
'
'      Major Version
'      Minor Version
'      Build Number
'      Revision
'
' You can specify all the values or you can default the Build and Revision Numbers
' by using the '*' as shown below:
' <Assembly: AssemblyVersion("1.0.*")>

<Assembly: AssemblyVersion("1.0.1.26")>
<Assembly: AssemblyFileVersion("1.0.1.26")>
```

The structure of this file is specific to .Net, so you should be able to find a reference copy in your source folders for your plugin.

If you aren't using Visual Basic as your language, the file may look different.

You will want to be sure that your AssemblyTitle correctly reflects the name of your plugin.

OFFICIALCHARACTERSHEET.VB

Your source file will be named as you name it; the filename here is just an example. You should always try to name your assemblies so that they aren't likely to conflict with any other plugin that might get loaded.

The code shown here includes just stubs of the various procedures you must implement. The full source code for a couple different plugins is in your GCA installation folder, in the \plugins\ subfolder.

Note that you will probably have to specify additional Imports (or your language's equivalent) because, once again, the compiler won't have any of the work done for it that Visual Studio often does for you.

```
'These aren't usually required Imports within Visual Studio,
‘but have to be included here now because the plugin compiler
‘doesn't make these associations automatically.
Imports Microsoft.VisualBasic
Imports System
Imports System.Collections
Imports System.Collections.Generic
Imports System.Diagnostics

'Everything from here and below is normal code and Imports and such, just as it is
'when developing within Visual Studio for VB projects.
Imports GCA5Engine
Imports C1.C1Preview
Imports System.Drawing
Imports System.Reflection

'Any such DLL needs to add References to:
'
'   System.Drawing  (System.Drawing; v4.X)
'   C1.C1Report.4   (ComponentOne Reports; v4.X)
'   GCA5Engine
'   GCA5.Interfaces.DLL
'
'in order to work as a print sheet.

Public Class OfficialCharacterSheet
    Implements GCA5.Interfaces.IPrinterSheet

    Public Event RequestRunSpecificOptions(sender As GCA5.Interfaces.IPrinterSheet, e As
            GCA5.Interfaces.DialogOptions_RequestedOptions) Implements
            GCA5.Interfaces.IPrinterSheet.RequestRunSpecificOptions

    Public ReadOnly Property PluginName() As String Implements
            GCA5.Interfaces.IPrinterSheet.PluginName
       Get

       End Get
    End Property
    Public ReadOnly Property PluginDescription() As String Implements
            GCA5.Interfaces.IPrinterSheet.PluginDescription
       Get
```

```
    End Get
End Property
Public ReadOnly Property PluginVersion() As String Implements
        GCA5.Interfaces.IPrinterSheet.PluginVersion
    Get

    End Get
End Property
Public Sub UpgradeOptions(Options As GCA5Engine.SheetOptionsManager) Implements
        GCA5.Interfaces.IPrinterSheet.UpgradeOptions

End Sub
Public Sub CreateOptions(Options As GCA5Engine.SheetOptionsManager) Implements
        GCA5.Interfaces.IPrinterSheet.CreateOptions

End Sub
Public Function GeneratePrinterDocument(GCAParty As Party, PageSettings As
        C1PageSettings, Options As GCA5Engine.SheetOptionsManager) As
        C1.C1Preview.C1PrintDocument Implements
        GCA5.Interfaces.IPrinterSheet.GeneratePrinterDocument

End Function
End Class
```

Required Interfaces

To create a sheet plugin that GCA can use, you must implement either the IPrinterSheet or the IExportSheet Interface. To create a Unified View box plugin, you must implement the IUnifiedViewBox Interface. GCA looks for these classes when loading plugins, to determine how to manage them.

To create a printer sheet, for example, you implement the IPrinterSheet Interface (which includes the elements from the IBasicSheet Interface, since the Sheet plugins descend from IBasicPlugin). You can see this in the stub example on the previous page, where the `Implements` keyword is used to tell the compiler which parts of the class implement which elements of the Interface.

Remember that the Interfaces provide templates for the features that your plugin must implement, exactly as they are shown. You cannot change the order of parameters, or specify different parameter or return types, because then the Interface is not being implemented, and the compilation will fail.

IBasicPlugin

Provides the required basic Interface all other plugins will build on.

```
Public Interface IBasicPlugin

    ReadOnly Property PluginName() As String
    ReadOnly Property PluginDescription() As String
    ReadOnly Property PluginVersion() As String

    Sub UpgradeOptions(Options As GCA5Engine.SheetOptionsManager)
    Sub CreateOptions(Options As GCA5Engine.SheetOptionsManager)

End Interface
```

Properties

Name

```
ReadOnly Property PluginName() As String
```

The name for your plugin. DO NOT use a colon : since that is used to separate print/export sheet names from profile names in the user's Sheet Options.

Description

```
ReadOnly Property PluginDescription() As String
```

A short description of your plugin.

Version

```
ReadOnly Property PluginVersion() As String
```

Your version number for your plugin.

Procedures

UpgradeOptions

```
Sub UpgradeOptions(Options As GCA5Engine.SheetOptionsManager)
```

This is called only when needed, such as when loading options for the plugin for the first time during a session, or when importing settings from a different profile.

This allows a plugin to evaluate the current set of options and to upgrade them to newer versions if necessary.

Parameters

- `Options As GCA5Engine.SheetOptionsManager`

  The SheetOptionsManager that currently contains the Options information for the plugin.

CreateOptions

```
Sub CreateOptions(Options As GCA5Engine.SheetOptionsManager)
```

This is the routine that creates and initializes the options for the plugin.

Parameters

- `Options As GCA5Engine.SheetOptionsManager`

  The SheetOptionsManager that currently contains the Options information for the plugin.

ICalledPlugin

Provides the required Interface for all plugins that handle their own UI and functionality, being called from the Launch menu. These plugins may either do their job and quit, or may stick around and work alongside the primary GCA application.

```
Public Interface ICalledPlugin
    Inherits IBasicPlugin

    Event CharacterChanged(Character As GCA5Engine.GCACharacter, sender As
IBasicPlugin)
    Event ListChanged(TraitType As GCA5Engine.TraitTypes, sender As IBasicPlugin)
    Event ShowOptions(sender As ICalledPlugin)
    Event ShutDown(sender As IBasicPlugin)

    Sub PartyChanged(Party As GCA5Engine.Party)
    Sub RefreshColors()
    Sub RefreshDisplay()
    Sub SetOptions(Options As GCA5Engine.SheetOptionsManager)
    Sub Start(Party As GCA5Engine.Party)
End Interface
```

Inherits

```
Inherits IBasicPlugin
```

Built on the basic plugin, so all its properties and procedures exist here as well.

Events

CharacterChanged

```
Event CharacterChanged(Character As GCA5Engine.GCACharacter, sender As ICalledPlugin)
```

Raise this event if something about the character data you are working on has changed, and the data is not related to a trait list (such as name, race, appearance, etc.).

Parameters

- `Character As GCA5Engine.GCACharacter`

  The character that has been changed.

- `sender As IBasicPlugin`

  The plugin that is raising this event.

ListChanged

```
Event ListChanged(TraitType As GCA5Engine.TraitTypes, sender As IBasicPlugin)
```

Raise this event if a certain type of trait list for the active character may have been changed or modified by your control (items added, removed; parents/children changed, etc.).

This event should be limited to changes in the structure of a list, such as traits added or removed, parent/child relationships changed, and so forth.

Parameters

- `TraitType As GCA5Engine.TraitTypes`

  The list that has changed from those specified in the GCA5.TraitTypes Enum.

- `sender As IBasicPlugin`

  The plugin that is raising this event.

ShowOptions

```
Event ShowOptions(sender As ICalledPlugin)
```

Raise this event when you want to display the Options dialog for this plugin. After the user closes the Options dialog, GCA will call SetOptions.

Parameters

- `sender As ICalledPlugin`

  The plugin that is raising this event.

ShutDown

```
Event ShutDown(sender As IBasicPlugin)
```

Raise this event when your plugin has completed its work and needs to be cleaned up after.

You must always raise this event when done working, or the instance of the control will not be able to be cleaned up, and the Launch entry for it will not be re-enabled.

Parameters

- `sender As IBasicPlugin`

  The plugin that is raising this event.

PROCEDURES

PartyChanged

```
Sub PartyChanged(Party As GCA5Engine.Party)
```

When the party of characters loaded in GCA changes, this procedure is called to let your plugin know which ones are now available, and which character is now currently active.

Parameters

- `Party As GCA5Engine.Party`

  A Party object that contains the characters currently available in GCA.

RefreshColors

```
Sub RefreshColors()
```

The system colors have changed. Redraw any dialog using the new colors, if you use the system colors.

RefreshDisplay

```
Sub RefreshDisplay()
```

General refresh request when something on the character has changed. You can check Char.ListChanged(ItemType)=True to see if a list that affects your plugin has changed. You can check Trait.Dirty=True to see if a particular trait appears to have changed.

SetOptions

```
Sub SetOptions(Options As GCA5Engine.SheetOptionsManager)
```

This is called when the plugin is initialized, or after the user closes the Options dialog that you requested with the ShowOptions event.

Parameters

- `Options As GCA5Engine.SheetOptionsManager`

  The SheetOptionsManager that currently contains the Options information for the plugin.

Start

```
Sub Start(Party As GCA5Engine.Party)
```

This is the procedure that is called to begin operations for your plugin; the Sub Main() if you will. This is where you'd instantiate any dialog required or other startup processes. Start() is only called once, when launched, and everything else is handled through the other calls.

Parameters

- `Party As GCA5Engine.Party`

  A Party object that contains the characters currently available in GCA.

IExportSheet

Provides the required Interface for all plugins that want to export to a file. Many of these are the same as for IPrinterSheet.

```
Public Interface IExportSheet
    Inherits IBasicPlugin

    Event RequestRunSpecificOptions(sender As IExportSheet, e As
            DialogOptions_RequestedOptions)

    Function GenerateExport(Party As GCA5Engine.Party, TargetFilename As String, Options As
            GCA5Engine.SheetOptionsManager) As Boolean
    Function PreferredFilterIndex() As Integer
    Function PreviewOptions(Options As GCA5Engine.SheetOptionsManager) As Boolean
    Function SupportedFileTypeFilter() As String
End Interface
```

Inherits

```
Inherits IBasicPlugin
```

Built on the basic plugin, so all its properties and procedures exist here as well.

Events

RequestRunSpecificOptions

```
Event RequestRunSpecificOptions(sender As IExportSheet, e As
        DialogOptions_RequestedOptions)
```

Raise this event if there are specific options you need from the user before things can proceed. This should happen after GenerateExport has been called, so you have Options and TargetFilename already set, and can create a new set of Options specific to this Event.

Parameters

- `sender As IExportSheet`

  The sheet plugin that is raising this event.

- `e As DialogOptions_RequestedOptions`

  The dialog options class that currently contains the RunSpecificOptions information for the sheet, which the user may change as needed. Canceled will be True when returned if the user canceled out of the dialog, in which case exporting should be aborted.

Procedures

GenerateExport

```
Function GenerateExport(Party As GCA5Engine.Party, TargetFilename As String, Options As
        GCA5Engine.SheetOptionsManager) As Boolean
```

This is the function that is called to generate the export file.

Parameters

- `Party As GCA5Engine.Party`

  A Party object that contains the characters available for printing.

- `TargetFilename As String`

  The filename (with fully qualified path) that the user specified as the desired export filename.

- `Options As GCA5Engine.SheetOptionsManager`

  The SheetOptionsManager that currently contains the Options information for the sheet.

Returns

- `Boolean`

  `True` if successful, `False` if not.

PreferredFilterIndex

```
Function PreferredFilterIndex() As Integer
```

Given the filter specified in SupportedFileTypeFilter(), this is the 0-based index into that filter list for the filter that should be selected by default.

Returns

- `Integer`

  The 0-based index into the SupportedFileTypeFilter() list for the filter that should be selected by default.

PreviewOptions

This function is called after options are loaded, to give the sheet a chance to do any unusual housekeeping before the Filter functions are called. Return `False` only if the export should be canceled.

Parameters

- `Options As GCA5Engine.SheetOptionsManager`

  The SheetOptionsManager that currently contains the Options information for the sheet.

Returns

- `Boolean`

  Return `True` if everything checks out. Return `False` only if the export should be canceled.

SupportedFileTypeFilter

```
Function SupportedFileTypeFilter() As String
```

Specifies the file types that this Export can write, by specifying a filter as used in the File dialog:

```
"GCA Character Files (*.gca5;*.gca4)|*.gca5;*.gca4|GCA5 files (*.gca5)|*.gca5|GCA4 files (*.gca4)|*.gca4|All files (*.*)|*.*"
```

If your exporter supports more than one file type you should check the extension on the TargetFilename in the GenerateExport function to be sure to export the correct type.

Returns

- `String`

  The string used by the File dialog to specify the types of files that are supported.

IPrinterSheet

Provides the required Interface for all plugins that want to output to the printer.

```
Public Interface IPrinterSheet
    Inherits IBasicPlugin

    Event RequestRunSpecificOptions(sender As IPrinterSheet, e As
            DialogOptions_RequestedOptions)

    Function GeneratePrinterDocument(Party As GCA5Engine.Party, PageSettings As
            C1.C1Preview.C1PageSettings, Options As GCA5Engine.SheetOptionsManager) As
            C1.C1Preview.C1PrintDocument
End Interface
```

Inherits

```
Inherits IBasicPlugin
```

Built on the basic plugin, so all its properties and procedures exist here as well.

Events

RequestRunSpecificOptions

```
Event RequestRunSpecificOptions(sender As IPrinterSheet, e As
        DialogOptions_RequestedOptions)
```

Raise this event if there are specific options you need from the user before things can proceed. This should happen after GeneratePrinterDocument has been called, so you have Options and PageSettings already set, and can create a new set of Options specific to this Event.

Parameters

- `sender As IPrinterSheet`

  The sheet plugin that is raising this event.

- `e As DialogOptions_RequestedOptions`

  The dialog options class that currently contains the RunSpecificOptions information for the sheet, which the user may change as needed. Canceled will be True when returned if the user canceled out of the dialog, in which case exporting should be aborted.

Procedures

GeneratePrinterDocument

```
Function GeneratePrinterDocument(Party As GCA5Engine.Party, PageSettings As
        C1.C1Preview.C1PageSettings, Options As GCA5Engine.SheetOptionsManager) As
        C1.C1Preview.C1PrintDocument
```

This is the function that is called to generate the Printer Document that will be printed or previewed.

We are using the ComponentOne C1PrintDocument as our document.

Parameters

- `Party As GCA5Engine.Party`

  A Party object that contains the characters available for printing.

- `PageSettings As C1.C1Preview.C1PageSettings`

  A System.Drawing.Printing.PageSettings object that contains page information.

- `Options As GCA5Engine.SheetOptionsManager`

  The SheetOptionsManager that currently contains the Options information for the sheet.

Returns

- `C1.C1Preview.C1PrintDocument`

  The formatted and complete print dccument.

IUnifiedViewBox

Provides the required interface for all plugins that work like controls on the Unified View.

```
Public Interface IUnifiedViewBox
```

```
    Inherits IBasicPlugin

    Event CharacterChanged(sender As IUnifiedViewBox)
    Event ListChanged(TraitType As GCA5Engine.TraitTypes, sender As IUnifiedViewBox)
    Event TraitsSelected(sender As IUnifiedViewBox)
    Event SpanChange(columns As Integer, sender As IUnifiedViewBox)
    Event ShowOptions(sender As IUnifiedViewBox)

    Property BorderWidth As Integer
    Property Character As GCA5Engine.GCACharacter
    Property ColumnSpan As Integer
    Property InstanceKey As String
    Property TraitType As GCA5Engine.TraitTypes

    Sub ClearSelection()
    Function GetSelectedTraits() As GCA5Engine.SortedTraitCollection
    Sub RebuildDisplay()
    Sub RefreshColors()
    Sub RefreshDisplay()
    Sub SetOptions(Options As GCA5Engine.SheetOptionsManager)
    Sub StartupMode(value As String)
    Function StartupModes() As String()
    Function StartupModeValue(value As String) As String
    Sub UpdateChangedLists()
    Sub UpdateDirtyItems()
```

```
End Interface
```

Inherits

```
Inherits IBasicPlugin
```

Built on the basic plugin, so all its properties and procedures exist here as well.

Events

CharacterChanged

```
Event CharacterChanged(sender As IUnifiedViewBox)
```

Raise this event if something about the character data you are working on has changed, and the data is not related to a trait list (such as name, race, appearance, etc.).

Parameters

- `sender As IUnifiedViewBox`

  The plugin that is raising this event.

ListChanged

```
Event ListChanged(TraitType As GCA5Engine.TraitTypes, sender As IUnifiedViewBox)
```

Raise this event if a certain type of trait list may have been changed or modified by your control (items added, removed; parents/children changed, etc.).

This event should be limited to changes in the structure of a list, such as traits added or removed, parent/child relationships changed, and so forth.

Parameters

- `TraitType As GCA5Engine.TraitTypes`

  The list that has changed from those specified in the GCA5.TraitTypes Enum.

- `sender As IPrinterSheet`

  The plugin that is raising this event.

TraitsSelected

```
Event TraitsSelected(sender As IUnifiedViewBox)
```

Raise this event when the user has selected one or more traits in the control. GCA will call GetSelectedTraits() to get the affected traits when it's ready for them.

Parameters

- `sender As IUnifiedViewBox`

  The plugin that is raising this event.

SpanChange

```
Event SpanChange(columns As Integer, sender As IUnifiedViewBox)
```

Raise this event when the control wants to change the number of columns that it spans.

Parameters

- `columns As Integer`

  The number of columns to span.

- `sender As IUnifiedViewBox`

  The plugin that is raising this event.

ShowOptions

```
Event ShowOptions(sender As IUnifiedViewBox)
```

Raise this event when you want to display the Options dialog for this control. After the user exits, GCA will call SetOptions.

Parameters

- `sender As IUnifiedViewBox`

  The plugin that is raising this event.

Properties

BorderWidth

```
Property BorderWidth As Integer
```

The name for your plugin. DO NOT use a colon : since that is used to separate print/export sheet names from profile names in the user's Sheet Options.

Character

```
Property Character As GCA5Engine.GCACharacter
```

The character to use for all data in the control.

ColumnSpan

```
Property ColumnSpan As Integer
```

The number of columns spanned by the control. This is more for reference in case you want to offer to change it in the gear menu, since this refers to the columns of the Unified View which handles setting your size.

InstanceKey

```
Property InstanceKey As String
```

GCA needs a reliable way to keep track of the controls, especially for ones that may have multiple instances. This is a unique key string for each instance of all controls. Your control should not change this value.

The InstanceKey is also used to help track what Options go with which control.

TraitType

```
Property TraitType As GCA5Engine.TraitTypes
```

If the control allows traits to be selected, and raises the TraitsSelected event, it needs to be able to return the type of trait selected since that affects some of the UI features. You can return 0 if you don't handle traits as such.

Procedures

ClearSelection

```
Sub ClearSelection()
```

Deselect all the selected items in the box.

GCA will call this for a variety of reasons, but the most common is if the user clicked into a different box.

GetSelectedTraits

```
Function GetSelectedTraits() As GCA5Engine.SortedTraitCollection
```

Return a GCA5Engine.SortedTraitCollection of the items that are currently selected in the box. Return a 0-count collection if no traits are selected.

Returns

- `GCA5Engine.SortedTraitCollection`

  The SortedTraitCollection that contains all the traits currently selected in your control.

RebuildDisplay

```
Sub RebuildDisplay()
```

Reconstruct yourself entirely, usually because the active character or view has changed.

RefreshColors

```
Sub RefreshColors()
```

The system colors have changed. Redraw the control using the new colors, if you use the system colors.

RefreshDisplay

```
Sub RefreshDisplay()
```

General refresh request when something on the character has changed that isn't trait specific. If you only deal with traits, you probably don't have to handle this.

SetOptions

```
Sub SetOptions(Options As GCA5Engine.SheetOptionsManager)
```

This is called when the plugin is initialized, or any time the user changes plugin options.

Parameters

- `Options As GCA5Engine.SheetOptionsManager`

  The SheetOptionsManager that currently contains the Options information for the plugin.

StartupMode

```
Sub StartupMode(value As String)
```

After the object is instantiated, it will get one entry from the StatupModes string array to tell it what to be. If StartupModes() returns nothing or empty, this won't be called.

For example, if you have two possible modes, Melee or Ranged, you might have StartupModes return a 2-value array of {"Melee", "Ranged"}. GCA will then instantiate two copies of your plugin, each of which will be passed a different one of these values.

Parameters

- `value As String`

  The string value that tells the control how it should initialize itself.

StartupModes

```
Function StartupModes() As String()
```

Returns an Array of Strings, the Count of which determines how many of this object will be created (but at least 1), and each Value will be given to each subsequent object's StartupMode()

Returns

- `String()`

  The array of strings that contains the various instantiation values for your plugin. Return Nothing if your control only needs one instance.

StartupModesValue

```
Function StartupModeValue(value As String) As String
```

Returns a String that represents the startup mode for the given string from the StartupModes Array.

Should be short and sweet, but each possible value from StartupModes should return a unique string since these are used to create the unique InstanceKeys.

The returned value here may be different from the values used in StartupModes!

Parameters

- `value As String`

  The string value that tells the control how it should initialize itself, as found in the array returned by StartupModes.

Returns

- `String`

  The unique value that identifies this plugin based on the value given.

UpdateChangedLists

```
Sub UpdateChangedLists()
```

Called when one or more TraitLists have changed (items added, removed, etc.). You can check Char.ListChanged(ItemType)=True to see if a list that affects your plugin has changed.

UpdateDirtyItems

```
Sub UpdateDirtyItems()
```

Called when Traits within certain TraitLists have become dirty (values changed). You can check Trait.Dirty=True to see if a trait that affects your plugin has changed.

Objects

ArmorLayer

The ArmorLayer class is used for calculation of coverage for a particular body part. Each layer contains information about what goes into that layer of protection, and in the end they're combined as best as possible to get a simpler display value.

This object is defined within the BodyItem class, as the BodyItems are used by the LoadOut to create the whole picture of protection.

```
    Public Class ArmorLayer
        Property ArmorItem As New GCATrait
        Property DB As String = "0"
        Property DR As String = "0"

        Property Deflect As Integer = 0
        Property Fortify As Integer = 0
        Property ApplyThisDeflect As Boolean = False
        Property ApplyThisFortify As Boolean = False

        Property IsFlexible As Boolean = False
        Property IsInnateDRValue As Boolean = False
        Property CountAsLayer As Boolean = True
        Property WaitListed As Boolean = False
        Property FootnoteSymbol As String = ""
    End Class
```

Body

Contains and manages the various body parts using BodyItem objects. This is basically a custom Collection of BodyItems.

```
Public Class Body
    Implements IEnumerable

    ReadOnly Property ErrorCode As Integer

    Public Sub Add(ByVal newItem As BodyItem, Optional ByVal Key As String = "")
    Public Function Count() As Integer
    Public Sub Clear()
    Public Function Item(ByVal Index As String) As BodyItem
    Public Function Item(ByVal Index As Integer) As BodyItem

    Public Sub Remove(ByVal Index As String)
    Public Sub Remove(ByVal Index As Integer)

    Public Sub New()
    Public Sub CopyTo(ByVal BodyCopy As Body)

    ''' <summary>
    ''' This routine is called by a contained BodyItem body part when a value for it has
        changed. This is done so that body parts that are components of larger groups
        (such as LeftArm being a part of Arms) can be updated with values inserted into
        those larger groups.
    ''' </summary>
    ''' <param name="BodyPart"></param>
    ''' <remarks></remarks>
    Public Sub PartChanged(ByVal BodyPart As BodyItem)

    Public Function GetEnumerator() As System.Collections.IEnumerator Implements
        System.Collections.IEnumerable.GetEnumerator
    Public Function XMLRead(Reader As XmlReader) As Boolean
    Public Sub XMLWrite(Writer As XmlWriter)
```

BodyItem

Each piece of a body that can be provided protection by armor or natural DR.

```
Public Class BodyItem
    ''' <summary>
    ''' ArmorLayer class is used for calculation of coverage for a particular body part.
    ''' Each layer contains info about what goes into that layer of protection,
    ''' and in the end they're combined as best as possible to get a simpler
    ''' display value.
    ''' </summary>
    Public Class ArmorLayer

    '** Used Locally, not saved. **

    ''' <summary>
    ''' Collection of ArmorLayer objects
    ''' </summary>
    ''' <value></value>
    ''' <returns></returns>
    ''' <remarks></remarks>
    Property ArmorLayers As Collection

    ''' <summary>
    ''' Collection of ArmorLayer objects.
    ''' These are items that aren't obvious when combining
    ''' armor values, so they're set aside separately.
    ''' </summary>
    ''' <value></value>
    ''' <returns></returns>
    ''' <remarks></remarks>
    Property WaitList As Collection

    ''' <summary>
    ''' When more than one exists, but only the best can be used
    ''' </summary>
    ''' <value></value>
    ''' <returns></returns>
    ''' <remarks></remarks>
    Property ThereAreUnstackedFortifyItems As Boolean = False

    ''' <summary>
    ''' When more than one exists, but only the best can be used
    ''' </summary>
    ''' <value></value>
    ''' <returns></returns>
    ''' <remarks></remarks>
    Property ThereAreUnstackedDeflectItems As Boolean = False

    '** End Local only block **

    ''' <summary>
    ''' Name of the body part
```

```
    ''' </summary>
    ''' <value></value>
    ''' <returns></returns>
    ''' <remarks></remarks>
    Property Name As String = ""

    ''' <summary>
    ''' This is so we can have multiple body types loaded at one time,
    ''' so the Cat might be human, bipedal, quadruped, etc.
    ''' Cat and Name may be combined for unique keys in collections
    ''' </summary>
    ''' <value></value>
    ''' <returns></returns>
    ''' <remarks></remarks>
    Property Cat As String = ""

    ''' <summary>
    ''' What its location might be, such as Arms to affect arms,
    ''' Hands to affect Hands, Head to affect skull, face, whatever.
    ''' This Group is what determines what's affected by what.
    ''' </summary>
    ''' <value></value>
    ''' <returns></returns>
    ''' <remarks></remarks>
    Property Group As String = ""

    ''' <summary>
    ''' The innate Base DB value before armor or anything else is added
    ''' </summary>
    ''' <value></value>
    ''' <returns></returns>
    ''' <remarks></remarks>
    Property BaseDB As String = "0"

    ''' <summary>
    ''' The innate Base DR value before armor or anything else is added
    ''' </summary>
    ''' <value></value>
    ''' <returns></returns>
    ''' <remarks></remarks>
    Property BaseDR As String = "0"

    ''' <summary>
    ''' The innate Base HP value before armor or anything else is added
    ''' </summary>
    ''' <value></value>
    ''' <returns></returns>
    ''' <remarks></remarks>
    Property BaseHP As String = "0"

    ''' <summary>
    ''' True or False as to whether this part is displayed
    ''' </summary>
```

```
    ''' <value></value>
    ''' <returns></returns>
    ''' <remarks></remarks>
    Property Display As Boolean = False 'true or false

    ''' <summary>
    ''' x coord for top left of edit box
    ''' </summary>
    ''' <value></value>
    ''' <returns></returns>
    ''' <remarks></remarks>
    Property PosX As Single = 0

    ''' <summary>
    ''' y coord for top left of edit box
    ''' </summary>
    ''' <value></value>
    ''' <returns></returns>
    ''' <remarks></remarks>
    Property PosY As Single = 0

    ''' <summary>
    ''' Width of the edit box
    ''' </summary>
    ''' <value></value>
    ''' <returns></returns>
    ''' <remarks></remarks>
    Property Width As Single = 0

    ''' <summary>
    ''' Height of the edit box
    ''' </summary>
    ''' <value></value>
    ''' <returns></returns>
    ''' <remarks></remarks>
    Property Height As Single = 0

    ''' <summary>
    ''' True to display expanded in properties, False to be collapsed
    ''' </summary>
    ''' <value></value>
    ''' <returns></returns>
    ''' <remarks></remarks>
    Property Expanded As Boolean = False

    ''' <summary>
    ''' Number of armor layers on this part
    ''' </summary>
    ''' <value></value>
    ''' <returns></returns>
    ''' <remarks></remarks>
    Property Layers As Integer = 0
```

```
    ''' <summary>
    ''' Body that owns this BodyItem
    ''' </summary>
    ''' <value></value>
    ''' <returns></returns>
    ''' <remarks></remarks>
    Property Owner As Body = Nothing

    ''' <summary>
    ''' Set the DB value for this part
    ''' </summary>
    ''' <value></value>
    ''' <returns></returns>
    ''' <remarks></remarks>
    Property DB As String

    ''' <summary>
    ''' Set the DR value for this part
    ''' </summary>
    ''' <value></value>
    ''' <returns></returns>
    ''' <remarks></remarks>
    Property DR As String

    ''' <summary>
    ''' Set the HP value for this part
    ''' </summary>
    ''' <value></value>
    ''' <returns></returns>
    ''' <remarks></remarks>
    Property HP As String

    ''' <summary>
    ''' Set to True when changing values so that altering the values won't trigger a call
         to Owner.PartChanged()
    ''' </summary>
    ''' <value></value>
    ''' <returns></returns>
    ''' <remarks></remarks>
    Property Freeze As Boolean

    ''' <summary>
    ''' Copies values to Target
    ''' </summary>
    ''' <param name="Target"></param>
    ''' <remarks></remarks>
    Public Sub CopyTo(ByVal Target As BodyItem)

    Public Function XMLRead(Reader As XmlReader) As Boolean
    Public Sub XMLWrite(Writer As XmlWriter)
End Class
```

DialogOptions_RequestedOptions

This is the class used to specify the options for RequestRunSpecificOptions, and to track if the user Canceled out or not.

```
Public Class DialogOptions_RequestedOptions
    Public Property Canceled As Boolean = False
    Public Property RunSpecificOptions As GCA5Engine.SheetOptionsManager
End Class
```

Properties

Canceled

```
Public Property Canceled As Boolean = False
```

Set to `True` if the user clicks Cancel on the provided dialog.

RunSpecificOptions

```
Public Property RunSpecificOptions As GCA5Engine.SheetOptionsManager
```

The SheetOptionsManager specific to this object, to be provided to the user in a dialog when RequestRunSpecificOptions is triggered.

LayerItem

A LayerItem is used for tracking layers in a LoadOut if the user is using the UserOrderedLayers option.

```
Public Class LayerItem
    Public Property Item As GCATrait
    Public Property IsInnate As Boolean
    Public Property CountAsLayer As Boolean
    Public Property IsFlexible As Boolean
    Public Property FootnoteSymbol As String

    Public Sub New()
    Public Function XMLRead(Reader As XmlReader, MyOwner As GCACharacter) As Boolean
    Public Sub XMLWrite(Writer As XmlWriter)
End Class
```

LoadOut

A LoadOut is used for tracking all the items in a particular loadout. This includes all the items carried, with special handling for all items that provide protection.

```
Public Class LoadOut
    ''' <summary>
    ''' Calc values even if there are no armor items. This is useful for including DR and
        partial DR when the character has no actual armor otherwise.
    ''' </summary>
    ''' <remarks></remarks>
    Public AlwaysAutoCalcArmor As Boolean = True

    ''' <summary>
    ''' All the items that are in this loadout
    ''' </summary>
    ''' <remarks></remarks>
    Public Items As SortedTraitCollection

    ''' <summary>
    ''' The subset of items in the loadout that the user has *selected* as active armor.
    ''' </summary>
    ''' <remarks></remarks>
    Public ArmorItems As SortedTraitCollection

    ''' <summary>
    ''' Collection of LayerItems. This is the ArmorItems items in the order that the user
        wants to apply them as layers, plus some additional info such as flexibility of
        innate traits.
    ''' </summary>
    ''' <remarks></remarks>
    Public OrderedLayers As Collection 'OF LayerItems

    ''' <summary>
    ''' User layer ordering?
    ''' </summary>
    ''' <remarks></remarks>
    Public UserOrderedLayers As Boolean = False

    Public LayerSeparator As String = "+"

    ''' <summary>
    ''' The subset of items in the loadout that the user has *selected* as active shields.
    ''' </summary>
    ''' <remarks></remarks>
    Public ShieldItems As SortedTraitCollection

    ''' <summary>
    ''' for each ShieldItem, there should be a ShieldArc of none, left arm, right arm, back
    ''' </summary>
    ''' <remarks></remarks>
    Public ShieldArcs As Collection
```

```
    ''' <summary>
    ''' Since a loadout can have armor or shields, it can have protection, so it also needs
         a Body
    ''' </summary>
    ''' <remarks></remarks>
    Public Body As Body
    Public Property Owner As GCACharacter
    Public Property Name As String
    Public ReadOnly Property SafeName As String
    Public Property Weight As Single
    Public Property ShieldDB As Integer
    Public Property HexMask As String
    Public Property FacingDB(index As Integer) As Integer
    Public Property OverlappingShieldArcs As Boolean

    Public Sub New(SetName As String, SetOwner As GCACharacter)
    Public Sub New()

    Public Sub ClearFacings()
    Public Function EncLevel() As Integer

    Public Sub Calculate()
    Public Sub CopyTo(TargetCopy As LoadOut)
    Public Sub RebuildActiveArmorParts()
    Public Sub RebuildActiveShields()

    ''' <summary>
    ''' Returns True if the GCATrait, or any parent up the chain, is contained in this
         LoadOut
    ''' </summary>
    ''' <param name="curItem">GCATrait</param>
    ''' <returns>Boolean</returns>
    ''' <remarks></remarks>
    Public Function ContainsItemOrParent(curItem As GCATrait) As Boolean

    ''' <summary>
    ''' Removes the item with the given CollectionKey from the active lists of
    ''' ArmorItems, ShieldItems, ShieldArcs, and OrderedLayers.
    ''' </summary>
    ''' <param name="ItemCollectionKey">The collection key for a GCATrait item.</param>
    ''' <remarks></remarks>
    Public Sub ArmorItemsRemove(ItemCollectionKey As String)

    ''' <summary>
    ''' Adds the GCATrait item to the active lists of
    ''' ArmorItems, ShieldItems, ShieldArcs, and OrderedLayers,
    ''' as applicable.
    ''' </summary>
    ''' <param name="newItem">An item of type GCATrait.</param>
    ''' <remarks></remarks>
    Public Sub ArmorItemsAdd(newItem As GCATrait)

    ''' <summary>
```

```
    ''' This function returns a collection that includes all Items, or their children, that
         qualify for use as Armor.
    ''' </summary>
    ''' <returns></returns>
    ''' <remarks></remarks>
    Public Function AllPossibleArmorItems() As Collection

    ''' <summary>
    ''' This function returns True if the item, or a child, contributes Protection to the
         loadout.
    ''' </summary>
    ''' <param name="curItem"></param>
    ''' <returns></returns>
    ''' <remarks></remarks>
    Public Function ItemOrChildProvidesProtection(curItem As GCATrait) As Boolean

    Public Sub AddItem(AddItem As GCATrait)
    Public Sub RemoveItem(RemoveItem As GCATrait)
    Public Function XMLRead(Reader As XmlReader) As Boolean
    Public Sub XMLWrite(Writer As XmlWriter)

    Public Sub ChangeBodyType(NewBodyType As String)

    Public Overrides Function ToString() As String

End Class
```

Party

A Party is an object that provides a collection of all the GCACharacter objects currently loaded, plus the character that is currently in focus within GCA.

```
Public Class Party
    Public Characters As Collection
    Public Current As GCACharacter
End Class
```

Properties

Characters

```
Public Characters As Collection
```

The collection of all characters currently loaded in GCA, each as a GCACharacter object.

Current

```
Public Current As GCACharacter
```

The character with the current focus in GCA.

This is usually the character you will be expected to work with.

Helpful Objects

These classes may be helpful with your sheets. They are not required to create plugins. However, they are a part of the GCA codebase already available to you in GCA5Engine, so they are ready for you to use in your sheets.

FileWriter

This class is available to help you write output to a file. This allows you to output linearly in a simple fashion.

Note that the file is not actually written to disk until you call FIleClose().

```
Public Class FileWriter
    Public Enum WriteMode As Integer

    Public ReadOnly Property NewLine As String

    Public Function CurrentLineNumber() As Integer
    Public Sub Write(ByVal Text As String)
    Public Sub WriteLine()
    Public Sub WriteLine(ByVal TextLine As String)

    Public Sub FileClose()
    Public Sub FileOpen(ByVal FileName As String, Optional ByVal WriteMode As WriteMode =
            FileWriter.WriteMode.Overwrite)
End Class
```

Enumerations

WriteMode

```
Public Enum WriteMode As Integer
    Overwrite = 0
    Append = 1
End Enum
```

For specifying whether you want to overwrite the file or add to the end of it.

Properties

NewLine

```
Public ReadOnly Property NewLine As String
```

Provides a line terminator, which is defined as CR + LF.

Procedures

CurrentLineNumber

```
Public Function CurrentLineNumber() As Integer
```

The current number of lines in the buffer to be printed. Note that these lines may not be the same as File lines, since they're not counting lines separated by CR+LF, but all text 'bits' added.

Not sure how useful that number is, but its there.

Returns

- `Integer`

  The number of text blocks added to the buffer.

Write

```
Public Sub Write(ByVal Text As String)
```

Adds text to the file exactly as provided; does not add NewLine.

Parameters

- `ByVal Text As String`

  Text string to be printed.

WriteLine

```
Public Sub WriteLine()
Public Sub WriteLine(ByVal TextLine As String)
```

Two overloads. Prints a NewLine to the file, or a chunk of text with a NewLine appended.

Parameters

- `ByVal TextLine As String`

  Text string to be printed.

FileClose

```
Public Sub FileClose()
```

If the WriteMode selected when opening the FileWriter is Overwrite, then this creates the file and writes all the text into it, overwriting any existing file.

If the WriteMode selected when opening the FileWriter is Append, then this appends all the text to any existing file data, or creates the file if it doesn't exist and writes the text to it.

FileOpen

```
Public Sub FileOpen(ByVal FileName As String, Optional ByVal WriteMode As WriteMode =
        FileWriter.WriteMode.Overwrite)
```

Creates a write buffer and specifies the way the file should be written to disk when FileClose is called.

Parameters

- `ByVal FileName As String`

  The name of the output file. This should include a fully qualified path.

- `Optional ByVal WriteMode As WriteMode = FileWriter.WriteMode.Overwrite`

  Whether to Overwrite or Append to existing files. Creates a new file if none exists.

FootnoteManager

This class is available to help you manage footnotes within your output.

```
Public Enum FootnoteMarkerStyle As Integer
Public Enum FootnoteEnclosureStyle As Integer

Public Class FootnoteManager
```

```
    Public Property FootnoteStyle As FootnoteMarkerStyle

    Public Function Add(ByVal NewFootnote As String) As String
    Public Sub Clear()
    Public Function Count() As Integer
    Public Function Footnote(ByVal Index As Integer) As String
    Public Function FootnoteBlock(Optional ByVal Separator As String = vbCrLf, Optional
            ByVal MarkerEnclosure As FootnoteEnclosureStyle = FootnoteEnclosureStyle.None)
            As String
    Public Function FootnoteWithMarker(ByVal Index As Integer, Optional ByVal
            MarkerEnclosure As FootnoteEnclosureStyle = FootnoteEnclosureStyle.None) As
            String
    Public Function Symbol(ByVal Index As Integer) As String
```

```
End Class
```

Enumerations

FootnoteMarkerStyle

```
Public Enum FootnoteMarkerStyle As Integer
```

```
    Symbol = 0
    Numeric = 1
    Alpha = 2
```

```
End Enum
```

For specifying the footnote style.

FootnoteEnclosureStyle

```
Public Enum FootnoteEnclosureStyle As Integer
```

```
    None = 0
    Parens = 1
    Brackets = 2
    Braces = 3
```

```
End Enum
```

For specifying the type of enclosures around your footnote marker.

Properties

FootnoteStyle

```
Public Property FootnoteStyle As FootnoteMarkerStyle
```

Set this to one of the values from the FootnoteMarkerStyle enum.

`FootnoteMarkerStyle.Symbol` will use the Steve Jackson Games standard symbols, in order, as footnotes are added: *, †, ‡, §, ¶. If additional footnotes are added and all symbols have been assigned, then double markers will be used, then triple, and so forth as more footnotes are assigned.

`FootnoteMarkerStyle.Numeric` will use numbers, ascending as footnotes are added.

`FootnoteMarkerStyle.Alpha` will use the standard English capital letters, in order from A through Z, as footnotes are added. If additional footnotes are added and all symbols have been assigned, then double markers will be used, then triple, and so forth as more footnotes are assigned.

Procedures

Add

```
Public Function Add(ByVal NewFootnote As String) As String
```

This function adds the text for a footnote to the manager, and returns the symbol corresponding to the footnote.

If the NewFootnote is unique, a new symbol is generated and returned; if it's not, an existing one is returned.

Parameters

- `NewFootnote As String`

  The text for the new footnote. If the NewFootnote is unique, a new symbol is generated and returned; if it's not, an existing one is returned.

Returns

- `String`

  The symbol for the footnote.

Clear

```
Public Sub Clear()
```

Clears the FootnoteManager, removing all existing footnotes and restoring assigned symbols to the beginning.

Count

```
Public Function Count() As Integer
```

The number of footnotes currently in the footnote manager (1-based).

Returns

- `Integer`

  The number of footnotes.

Footnote

```
Public Function Footnote(ByVal Index As Integer) As String
```

Returns the footnote at the given Index.

Parameters

- `Index As Integer`

  The numeric index (1-based) of the footnote desired.

Returns

- `String`

  The requested footnote. If there is no such footnote, returns an empty string.

FootnoteBlock

```
Public Function FootnoteBlock(Optional ByVal Separator As String = vbCrLf, Optional ByVal
        MarkerEnclosure As FootnoteEnclosureStyle = FootnoteEnclosureStyle.None) As String
```

Gets a text block consisting of ALL the footnotes in the FootnoteManager. By default, each will be on a separate line (CR + LF delimited), each line beginning with the footnote symbol and a space, and with no enclosures.

Parameters

- `Optional ByVal Separator As String = vbCrLf`

  Optionally allows you to set the separator between the footnotes in the text block. Default is CR + LF.

- `Optional ByVal MarkerEnclosure As FootnoteEnclosureStyle = FootnoteEnclosureStyle.None`

  Optionally allows you to set the enclosure style for the footnote markers, if you'd like them enclosed.

Returns

- `String`

  The text block containing all the footnotes.

FootnoteWithMarker

```
Public Function FootnoteWithMarker(ByVal Index As Integer, Optional ByVal MarkerEnclosure
        As FootnoteEnclosureStyle = FootnoteEnclosureStyle.None) As String
```

Gets a piece of text consisting of the desired footnote prefixed with its marker symbol and a space. Marker is optionally enclosed in parentheses, braces, or brackets.

Parameters

- `ByVal Index As Integer`

  The numeric index (1-based) of the footnote desired.

- `Optional ByVal MarkerEnclosure As FootnoteEnclosureStyle = FootnoteEnclosureStyle.None`

  Allows you to set the enclosure style for the footnote markers if you'd like them enclosed.

Returns

- `String`

  The text of the footnote prefixed with the marker and a space.

Public Function Symbol(ByVal Index As Integer) As String

```
Public Function Symbol(ByVal Index As Integer) As String
```

Returns the footnote symbol corresponding to Index (1-based).

Parameters

- `ByVal Index As Integer`

  The numeric index (1-based) of the symbol desired.

GroupedTraitListBuilder

This class exists to allow you to easily generate the lists you need to support Trait List Grouping options.

```
Public Class GroupedTraitListBuilder
    Public Property Character As GCACharacter
    Public Property IncludeAllAttributes As Boolean = False
    Public ReadOnly Property GroupingOptions As TraitListGroupingOptions
    Public Property ItemType As TraitTypes
    Public Property OrderBy As String = ""
    Public Property ShowComponents As Boolean = False
    Public Property ShowHiddenTraits As Boolean = False

    Public Sub BuildGroupedTraits()
    Public Function GroupedTraits() As Dictionary(Of String, Collection)
    Public Function ValuesToGroupBy() As Collection
End Class
```

Properties

Character

```
Public Property Character As GCACharacter
```

The GCACharacter for which we're processing data.

IncludeAllAttributes

```
Public Property IncludeAllAttributes As Boolean = False
```

By default, only Attributes with a MainWin() tag value are included in the lists. Set this to True to include all Attributes, regardless of MainWin() setting.

GroupingOptions

```
Public ReadOnly Property GroupingOptions As TraitListGroupingOptions
```

If ItemType and Character are set, this returns a copy of the TraitListGroupingOptions object that is used to build the lists. The actual TraitListGroupingOptions are set in Options by the user for each Character.

ItemType

```
Public Property ItemType As TraitTypes
```

The TraitType for the list we'll be processing.

Set/Get one of GCA's TraitTypes enum values, or an integer equivalent.

OrderBy

```
Public Property OrderBy As String = ""
```

Set the ordering to be used, if any. This uses the TagOrderFormat used by the SortedTraitCollection.

The TagOrderFormat works like this:

```
tagname[+|-][$|#] , tagname2[+|-][$|#]
```

The various bits are:

- `tagname`

  This is the name of the tag to order by.

- `+`

  Sort ascending

- `-`

  Sort descending

- `$`

  Treat as text

- `#`

  Treat as numeric

You pick one each of the `+` or - and `$` or `#` bits.

To order by level and then by name, you might use this OrderBy:

```
OrderBy = "Level+#, name+$"
```

ShowComponents

```
Public Property ShowComponents As Boolean = False
```

By default, Component traits (those that are included in a racial template or meta-trait) are not included in the lists. Set this to `True` to include them.

ShowHiddenTraits

```
Public Property ShowHiddenTraits As Boolean = False
```

By default, Hidden traits are not included in the lists. Set this to True to include them.

PROCEDURES

BuildGroupedTraits

```
Public Sub BuildGroupedTraits()
```

Builds the ValuesToGroupBy and GroupedTraits data elements.

In other words, once your properties are set, this does all the work. Then the results can be retrieved through the GroupedTraits() and ValuesToGroupBy() functions.

Public Function GroupedTraits() As Dictionary(Of String, Collection)

```
Public Function GroupedTraits() As Dictionary(Of String, Collection)
```

Returns a Dictionary keyed by the ValuesToGroupBy values, each entry of which is a Collection of traits within the group.

Returns

- `Dictionary(Of String, Collection)`

  A Dictionary keyed by the ValuesToGroupBy strings, where each dictionary value is a Collection of traits that satisfied the requirements to be included in the group.

Public Function ValuesToGroupBy() As Collection

```
Public Function ValuesToGroupBy() As Collection
```

Returns the collection of values used to group the traits. These are strings representing the tags or category values into which the traits were grouped.

There will almost always be an entry with a value of “” (empty string) included, which will allow for a collection in the GroupedTraits dictionary for all traits not otherwise grouped.

Returns

- `Collection`

  A collection of string values.

ProtectionPaperDollSettings

Allows you to set features for the GetProtectionPaperDoll function.

```
Public Class ProtectionPaperDollSettings
    ''' <summary>
    ''' The character to use.
    ''' </summary>
    ''' <value>GCACharacter</value>
    ''' <returns>GCACharacter</returns>
    ''' <remarks></remarks>
    Public Property Character As GCACharacter

    ''' <summary>
    ''' The font to use for the text elements. Default is Microsoft Sans Serif, 8.25pt.
    ''' </summary>
    ''' <value>Drawing.Font</value>
    ''' <returns>Drawing.Font</returns>
    ''' <remarks></remarks>
    Public Property TextFont As Drawing.Font = New Drawing.Font("Microsoft Sans Serif",
        8.25)

    ''' <summary>
    ''' The color of the non-shaded text elements. Default is Black.
    ''' </summary>
    ''' <value>Drawing.Color</value>
    ''' <returns>Drawing.Color</returns>
    ''' <remarks></remarks>
    Public Property TextColor As Drawing.Color = Drawing.Color.Black

    ''' <summary>
    ''' The primary background color of the protection value boxes. Default is White.
    ''' </summary>
    ''' <value>Drawing.Color</value>
    ''' <returns>Drawing.Color</returns>
    ''' <remarks></remarks>
    Public Property BackColor As Drawing.Color = Drawing.Color.White

    ''' <summary>
    ''' The color of the text on the shaded background of the value boxes. Default is
        Black.
    ''' </summary>
    ''' <value>Drawing.Color</value>
    ''' <returns>Drawing.Color</returns>
    ''' <remarks></remarks>
    Public Property TextColorAlt As Drawing.Color = Drawing.Color.Black

    ''' <summary>
    ''' The color of the shading for the text headings. Default is LightGray.
    ''' </summary>
    ''' <value>Drawing.Color</value>
    ''' <returns>Drawing.Color</returns>
    ''' <remarks></remarks>
```

```
    Public Property ShadeColor As Drawing.Color = Drawing.Color.LightGray

    ''' <summary>
    ''' The color of the borders around the protection value boxes. Default is Black.
    ''' </summary>
    ''' <value>Drawing.Color</value>
    ''' <returns>Drawing.Color</returns>
    ''' <remarks></remarks>
    Public Property BorderColor As Drawing.Color = Drawing.Color.Black

    ''' <summary>
    ''' If True, the image won't show any DR boxes where DR=0, otherwise it will. Default
         is False.
    ''' </summary>
    ''' <value>Boolean</value>
    ''' <returns>Boolean</returns>
    ''' <remarks></remarks>
    Public Property DoNotShowDRZero As Boolean = False

    ''' <summary>
    ''' The width of the returned bitmap, in pixels. Use 0 for native size. Default is 0.
    ''' </summary>
    ''' <value>Integer</value>
    ''' <returns>Integer</returns>
    ''' <remarks></remarks>
    Public Property DesiredWidthInPixels As Integer = 0

    ''' <summary>
    ''' Include the inset graphic of the shield arcs. Default is True.
    ''' </summary>
    ''' <value>Boolean</value>
    ''' <returns>Boolean</returns>
    ''' <remarks></remarks>
    Public Property IncludeShieldArcs As Boolean = True
End Class
```

TraitListGroupingOptions

The object that contains the various option settings for the grouping that GCA supports in certain trait lists.

```
Public Enum TraitGroupingType As Integer

Public Class TraitListGroupingOptions
    Public Property AppliesTo As TraitTypes = TraitTypes.None
    Public Property GroupingType As TraitGroupingType = TraitGroupingType.None
    Public Property GroupsAtEnd As Boolean = False
    Public Property IncludeTagPartInHeader As Boolean = True
    Public Property SpecifiedTag As String = ""
    Public Property SpecifiedValuesList As String = ""
    Public Property SpecifiedValuesOnly As Boolean = False

    Public Sub CopyTo(Target As TraitListGroupingOptions)
    Public Function Matches(Target As TraitListGroupingOptions) As Boolean
    Public Function XMLRead(Reader As XmlReader) As Boolean
    Public Sub XMLWrite(Writer As XmlWriter)
End Class
```

Enumerations

Public Enum TraitGroupingType As Integer

```
Public Enum TraitGroupingType As Integer
    None = 0
    ByCategory = 1
    ByTag = 2
End Enum
```

For specifying the types of grouping that GCA supports.

Properties

Public Property AppliesTo As TraitTypes = TraitTypes.None

```
Public Property AppliesTo As TraitTypes = TraitTypes.None
```

Specifies the trait type to which this grouping option applies.

Set/Get one of GCA's TraitTypes enum values, or an integer equivalent.

Public Property GroupingType As TraitGroupingType = TraitGroupingType.None

```
Public Property GroupingType As TraitGroupingType = TraitGroupingType.None
```

Specifies grouping by category or tag.

Set/Get one of the TraitGroupingType enum values.

GroupsAtEnd

```
Public Property GroupsAtEnd As Boolean = False
```

When grouping only by specified categories, they will be shown at the front of the trait listing, unless you set this property to `True`, in which case they'll be at the end of the listing instead.

IncludeTagPartInHeader

```
Public Property IncludeTagPartInHeader As Boolean = True
```

If grouping by tag, include 'tag = ' in the grouping header.

SpecifiedTag

```
Public Property SpecifiedTag As String = ""
```

If grouping by tag, the tag by which to group.

SpecifiedValuesList

```
Public Property SpecifiedValuesList As String = ""
```

The comma separated list of cats or tag values to group by.

SpecifiedValuesOnly

```
Public Property SpecifiedValuesOnly As Boolean = False
```

Group only by a list of specified cats or tag values.

PROCEDURES

CopyTo

```
Public Sub CopyTo(Target As TraitListGroupingOptions)
```

Copies the values of this object to the Target object.

Parameters

- `Target As TraitListGroupingOptions`

  The object to copy properties into.

Matches

```
Public Function Matches(Target As TraitListGroupingOptions) As Boolean
```

Returns True if Target object has all the same property values as this object.

Parameters

- `Target As TraitListGroupingOptions`

  The object to copy properties into.

XMLRead

```
Public Function XMLRead(Reader As XmlReader) As Boolean
```

Reads the XML representation of this object from the given Reader.

Parameters

- `Reader As XmlReader`

  The XML reader providing the XML document.

XMLWrite

```
Public Sub XMLWrite(Writer As XmlWriter)
```

Writes the XML representation of this object with the given Writer.

Parameters

- `Writer As XmlWriter`

  The XML writer creating the XML document.

TraitTypes Enum

Provides a handy way to reference the numeric values used by GCA for the various types of lists tracked by trait type, without having to use numeric constants.

```
''' <summary>
''' The various types of traits supported by GCA.
''' </summary>
''' <remarks></remarks>
Public Enum TraitTypes As Integer
    Modifier = -1

    Other = 0
    None = 0

    Stats = 1
    Attributes = 1
    Languages = 2
    Cultures = 3
    Ads = 4
    Advantages = 4
    Perks = 5
    Disads = 6
    Disadvantages = 6
    Quirks = 7
    Features = 8
    Skills = 9
    Spells = 10
    Equipment = 11
    Packages = 12
    Templates = Packages
    LastItemType = Packages
End Enum
```

Note that Modifier is a special case, used internally by GCA. I can't think of any situation where you might need to use that with GCA data in your plugin.

Disorganized notes for Future Sections and Content

Things Writers of Plugins Should Know

Since release, and as of 5.0.197, there have been some important changes to be aware of:

*** Notes/Notes()**

As mentioned in the build notes, the purpose of the notes() tag on traits has been pretty muddled over time due to a variety of folks using it in ways that were not intended. That tag was meant to be used in a tiny space to show table notes references such as "[1,2]" for weapon or armor footnotes. Yes, notes() was a bad name for it, but it originated back when GCA did far less.

The notes() tag is now enforced as a mode-specific tag for the original intended purpose (and even that is now discouraged in favor of using itemnotes() instead), and any use of notes() that was intended by a user to be general item-related notes or commentary should be changed to usernotes() or included in the description(), instead.

*** Each trait's Description() and UserNotes() tags are probably RTF formatted strings.**

This may not work for you.

The new functions GetNotes(Optional ByVal AsPlainText As Boolean = True) As String and GetDescription(Optional ByVal AsPlainText As Boolean = True) As String allow for retrieving the usernotes() and description() tags of a trait, as normal RTF or as plain-text.

Line breaks (as LF characters) will be preserved.

*** The Notes property on GCATrait has been will now return usernotes() as plain-text.**

If you set it, it sets usernotes() with that text.

*** Character.Settings has an ApplyDBToActiveDefenses property.**

When TRUE, all active defense scores (block, parry, dodge) *will* include the DB value for the active shield (one equipped on an arm). When FALSE, those scores won't include the DB (which is the traditional way GCA has done it).

If your application needs the scores *not* to include the shield DB but this setting is TRUE, you can set the value to FALSE, call RecalculateAll(True, False), do your exporting, then reset things for the user by setting the value back to TRUE and calling RecalculateAll(True, False) before quitting.

You can get the current DB being applied in the DefenseBonus() As Integer or DB() As Integer functions. The applied DB may be 0 if no shield is equipped.

*** Each shield will have a charblockscore() tag.**

This contains the character's Block level when using that shield. This will include the DB for the shield, but if ApplyDBToActiveDefenses is TRUE, then the better of the two values (this shield's DB or equipped shield's DB) will be included.

*** GCA provides the VTTNotes() tag.**

This is per-trait as a way for the user to include VTT-specific notes or formulas, such as OTF expressions for Foundry. On an attack-mode level, GCA provides the mode-specific VTTModeNotes() tag for the same purpose.

*** I have added Public Function DamageDisplayText() As String to the Mode object.**

This function returns a string that represents the standard damage notation including damage, armor divisor, damage type, and radius. Example: 2d+1 (2) cut (2). You can also access this using Item.DamageModeTagItem() with "DamageDisplayText" as the tag, if you're accessing mode values that way. This should obviate the need for you to build damage strings yourself, and it will correctly show the new comma-separated mini-modes if those are used by a trait.

DO YOU HAVE ANY DOCUMENTATION ON THE GCACHARACTER.ITEMS LIST?

I don't believe I do at the moment. The Items list is a SortedTraitCollection (basically a typed/customized Collection) of all GCATraits. The ItemsByType list is a TraitsByTypeCollection (basically a typed Collection for SortedTraitCollection objects). If you access SortedTraitCollection items, you get back GCATraits, keyed by each trait's CollectionKey. If you access TraitsByTypeCollection you get back a SortedTraitCollection.

So, they only contain all the various traits (advantages through equipment and templates).

The GCACharacter also contains tons of other stuff, such as bonuses, categories, groups, campaign stuff, body stuff, etc.

ColorBlockSheet breaks things apart differently than OfficialCharacterSheet, so may be easier to parse for examples.

```
''' <summary>
''' The various types of traits supported by GCA.
''' </summary>
''' <remarks></remarks>
Public Enum TraitTypes As Integer
    Modifier = -1

    Other = 0
    None = 0

    Stats = 1
    Attributes = 1
    Languages = 2
    Cultures = 3
    Ads = 4
    Advantages = 4
    Perks = 5
    Disads = 6
    Disadvantages = 6
    Quirks = 7
    Features = 8
    Skills = 9
    Spells = 10
    Equipment = 11
    Packages = 12
```

```
    Templates = Packages
    LastItemType = Packages
End Enum
```

You can reference the TraitTypes Enum when using Character.TraitsByType to get back the specific SortedTraitCollection that you want to look at, such as when printing Advantages or Spells.

NOTE: I may change the order, insert new items, or otherwise mess up the numbers in the Enum. While that isn't likely to happen at the moment, it has happened in the past, so please do not use literal constants in your plugins.

How do I access Loadouts?

A LoadOut is an object, so to get the current loadout for a character, you would do something like this:

```
Dim MyCurrentLoadOut As LoadOut = MyChar.LoadOuts(MyChar.CurrentLoadout)
```

LoadOuts is a LoadOutManager object, which handles a collection of LoadOut objects. CurrentLoadout is a character property containing the currently selected loadout name.

There are three SortedTraitCollections in a LoadOut: Items provides access to all the traits in the loadout; ArmorItems provides access to all the traits in the loadout that provide protection, and ShieldItems provides access to the traits selected as active shields.

You could iterate any of these SortedTraitCollections using For Each:

```
For Each curItem In MyCurrentLoadOut.ArmorItems
```

If you want to see if an ArmorItem is also a ShieldItem, you should check it using the collection key:

```
If MyCurrentLoadOut.ShieldItems.Contains(curItem.CollectionKey) Then
```

Because every loadout can also specify a different set of items for protection, every loadout has its own Body object to handle that.

Users also have the option of manually ordering the way that their armor is layered. This is useful when using the nitty-gritty armor rules. You can check for it with the MyCurrentLoadOut.UserOrderedLayers property.

Even if the user selected UserOrderedLayers, they may not have assigned any layers to the current loadout, so you should also check that the OrderedLayers collection exists. If it does not, use the unordered method to reference the protection objects in the loadout.

```
Dim curLayer As LayerItem
Dim curItem As GCATrait

If Not MyCurrentLoadOut.UserOrderedLayers OrElse MyCurrentLoadOut.OrderedLayers Is Nothing
         Then
    'straight item list

    For Each curItem In MyCurrentLoadOut.ArmorItems
        'check if it's a shield, and in this case don't use the item if it is
        If Not MyCurrentLoadOut.ShieldItems.Contains(curItem.CollectionKey) Then

            'do stuff with the armor item
```

```
        End If
    Next
Else
    'layered order

    If MyCurrentLoadOut.OrderedLayers.Count > 0 Then
        For Each curLayer In MyCurrentLoadOut.OrderedLayers
            curItem = curLayer.Item

            'do stuff with the armor item

        Next
    Else
        ‘No applicable traits
    End If
End If
```

How do I show the graphic with the various body locations and armor?

GCA provides a function to get a bitmap image of the user's protection graphic, with the active armor locations printed on it:

```
Public Function GetProtectionPaperDoll(Settings As ProtectionPaperDollSettings) As
        Drawing.Bitmap
```

The Settings parameter allows you to adjust settings for the output. The character's current loadout is used to generate the image.

How do I access the body type, body plan, body parts, or Hit Table for the character?

This can get a bit confusing, because GCA handles Body as one thing, and Hit Table as another.

Every loadout has a Body and a Hit Table assigned to it. These are based on the character's DefaultBody and DefaultHitTable, but both the body and the hit table can be changed by the user to fit what applies for that loadout. It should be that the body type always matches the hit table being used, but there are sometimes valid reasons for that not to be the case (for example, a custom body may still use the same targetable hit locations as a known body type).

A Body is a named object with a full collection of all the body parts that might have DR and HP, and might get protection from traits and armor; plus the bits needed to handle all that.

A HitTable is a hit location table, which has a name and a bunch of named locations with penalties and notes for them.

Technically…

…hit location tables should be assigned by and to transforms. However, transforms are structurally collections of items, and don't do anything that needs to know what the hit locations are, while loadouts do need to know what the locations are to provide protection to them. So, each loadout is instead allowed to have a body, and therefore a body type with hit locations and a hit location table. Having that information in a transform would mean that every transform would have to have a set of loadouts, and since almost no characters need transforms, that was deemed too complicated, and I went with the current system.

So, exactly what you're doing will affect whether you need to reference the Body or the HitTable, but in most cases you'll want to get the currently active loadout before you do that.

If you just need a quick reference, you can use Char.BodyType to access the name of the body type (the selected Body for the character). However, that points to the active loadout, which might be nothing, and therefore this might be empty as well. You could also access Char.DefaultBodyType to get the name, which should always be *something*, but might be Humanoid when the user intended something else and simply never set the default (generally by setting the body for the **All Unassigned Items** virtual loadout).

The best way to access the current body type is to check the current loadout, and if the current loadout is nothing, get the **All Unassigned Items** loadout from GCA. From there, you'd use the Body for all the locations that exist. Or, if you want to fit it into a hit location table, you could use the HitTable to get those location names, and then check the Body for those specific locations.

Note: there are some exceptions in the naming of some hit location table entries versus how they're named in the body locations: hit location tables often use *hand*, *foot*, and *eye* while most body parts lists use *hands*, *feet*, and *eyes*.

This discrepancy came about due to data entry honoring how they're used in the source material instead of being concerned with our data integrity. Just so you know, if you use HitTable entries to search for locations in a Body, you'll need to account for this. (The CreateHitTableGrid routine in the Spring Bandit Sheet creates a hit location table that optionally prints DR values as well, and it uses an exception collection to correct for this, if you need an example.)