GURPS® Character Assistant v2 Data File Format Specification

31 March 1997

GURPS Character Assistant (GCA) is a product of Armin D. Sykes, published by Miser Software.

Contents

Purpose

This file will attempt to specify for you, as clearly as possible, the correct format for creating data files for use with v2.1 of GCA. The data files created with this specification are *not* compatible with v1.1 or earlier of GCA. While v1 data files can be read by GCA v2, those data files lack a great deal of information, and because of this, some functionality will be missing.

Some General Things First

Before we get into the overall structure of the data files, we should cover a few very general things about data files.

First, blank lines are usually ignored by GCA, except when reading the Author section of a data file.

Second, there are some characters which have special meaning to GCA throughout a data file:

Comments

* an asterisk at the beginning of a line denotes a comment, and the whole line will be ignored (except in the Author section). An asterisk in the middle of the line will be taken to be the multiplication symbol, not a comment.

// a pair of slashes together at any point in a line denotes a comment. Anything following the comment marker will be ignored (except in the Author section).

Comments have precedence over all other special characters, so anything following a comment will never be seen. Never put line continuation characters after a comment marker, because GCA will never see it.

Line Continuation

_ an underline at the end of a line denotes a line continuation. The following line will be taken and appended at the end of the current line, and the underline character will be dropped. Note that an underline in a comment is ignored, so you should make sure not to try continuing comments with this character.

, commas are used to separate tags (explained below). If you end a line with a comma, GCA will include the following line as part of the current line, negating the need to use an underline in these cases.

| the OR symbol is used only in needs( ) tags to separate one group of possible prerequisite items from another. It's mentioned here only because ending a line with this character works the same way as ending the line with a comma.

Restricted Characters

Because GCA uses certain characters to mean certain things internally, you should avoid using certain characters. These characters should be considered restricted:

```
( ) , ; "
```

You can often use these characters safely within most tags, but it is better to avoid them in most cases unless they are part of a tag being used in the specified manner. In particular, be careful when using (and), because having an uneven number of either one can cause GCA to hang when reading a data file.

Prefix Tags

Prefix Tags are special tags tacked on to the front of the name of an ad, disad, quirk, power, skill or spell that tells GCA exactly where to look for that particular item. Valid tags are:

```
AD:      for advantages
ADS:     for advantages
PO:      for powers
POWERS:  for powers
DI:      for disadvantages
DISADS:  for disadvantages
QU:      for quirks
QUIRKS:  for quirks
SK:      for skills
SP:      for spells
GR:      for groups
CL:      for classes
CO:      for colleges
```

It is to your advantage to use a Prefix Tag in all cases where you are referring to another item, since it will speed up how fast things are found. In most cases, Prefix Tags are *required*, not optional, so you should get in the habit of using them.

Math

In several areas in a data file, GCA will support the evaluation of mathematical expressions. Because of this, GCA must be able to see and recognize math characters for what they are. There are several characters that are recognized as math characters in these areas, and you should take special care. These characters are all math characters:

```
( ) + - * / < > =
```

If any of these characters appears within the name of an item that is being used in a math section, that item name must be enclosed within double quotes, including any prefix tag, if applicable. For example, if a weapon skill was to default from the skill Axe/Mace at -4, then the default( ) tag for that item should look like this:

```
default(“SK:Axe/Mace”-4)
```

Notice that the prefix tag of SK: is included within the quotes along with the name of the skill, but the rest of the math expression appears outside of the quotes.

Functions

There are several simple functions supported in places where math can be used. These functions are described here.

@MAX( ) and @MIN( )

These two functions simply return the highest or lowest value from a list of values. If appropriate, the values in the list may use math also, including references such as ‘prereq’ or ‘default’ if being used in a skill line. The use of these functions looks something like this (in an upto() tag, which supports math):

```
upto(@max(12, prereq-2, default-5))
```

What happens in this upto() tag is this: the function is looked at and everything in the list of items between the parenthesis for the function is evaluated. Then, the highest of those numbers is selected (lowest if you were using @min). So, if the level of the prerequisite skill for the item above was 15 and the level of the skill we were defaulting from was 15 also, the upto() tag would look like this to GCA after all of the calculations were made:

```
upto(@max(12, 13, 10))
```

So, the upto( ) limit on the item would be the highest value between the three numbers in the list, which is 13 in this case. If the @min function had been used instead, the upto( ) limit on the skill level would be 10 instead.

Note: any number of items can be used in the @max and @min items list, so long as they are all separated by commas. And, as you can see, math within each item of the list is supported as well. Remember that you must be sure to quote the names of items that use reserved math characters (see Math above), and if you want to use the points of a quoted item, include the ‘ pts’ part within the double quotes.

@IF( )

The @if function allows you to specify one of two possible results, depending on an evaluation. The syntax for the @if function is like this:

```
@IF(expression THEN result ELSE altresult)
```

Notice that because it is a function, the entire statement (except the @if part) is enclosed within parens. 'expression' is an expression, such as IQ > 10, a bit of math, or whatever. (See the new list of math symbols above!) 'result' and 'altresult' are values or math that results in a value. If 'expression' is any **non-zero** result, including a true evaluation (such as 3>2), then the 'result' is used as the result of the function, otherwise 'altresult' is used as the result of the

function. If there is no Else part, then 0 would be returned for any false, or zero, result of 'expression'. An example would be this upto() tag:

```
upto(@if(IQ > 10 then 10 else 5))
```

(watch those parens to make sure they match!) In this case, if the character's IQ is more than 10, upto() would be 10, otherwise it would be 5.

You can nest one @if function within another, but be very cautious when you do so, because it can be very easy to loose track of the parens, and thereby crash GCA when loading your data file because of unmatched pairs.

Overall File Structure

The data file is divided into several main sections: version, author, bonus, stats, ads, powers, disads, quirks, skills, spells, packages, groups, and mods. Each of these sections has a particular structure. Aside from the Version section, which must be the very first section (and line) in the data file, the order of the sections within the data file is not important.

Also, except for the Version section, the start of each section is marked by the name of the section enclosed by square brackets, like this:

```
[ADS]
```

When GCA sees a line in the data file that looks like this, it knows that it is starting a chunk of that kind of data. You do not need to include this header for any section in your data file that doesn't include data.

Each section is explained in more detail below. All examples within each section below are indented to make it easier to see what is part of the example. You should not bother to indent in your data files. If any of the explanation given below doesn't quite seem to make sense, you are encouraged to examine previously created data files to see how other people are doing things.

Tags

Tags are the basic structural item for most of the data sections in a GCA data file. The basic format of a tag is this:

```
tagname(tag info)
```

The 'tagname' specifies what information is being listed, and the 'tag info' contains the information that is being specified. The parenthesis before and after the 'tag info' are required, and the opening parenthesis must come immediately after the 'tagname', and should be considered a part of the 'tagname'. Do not include a space between the 'tagname' and the '(', or GCA will not recognize the tag!

General Structure Diagram

Here is a simple diagram of what the general structure of a data file looks like.

```
Version=2

[Author]
Filename
Date of creation/modification
File/worldbook copyright information
File requirements information
General copyright/license information

[Bonus]
<Item=Score>
+Bonus SK:Skill getting bonus

[Ads]
<category>
ad item name (name ext), cost, tag1(tag1 info), tag2(tag2 info)

[Powers]
<category>
power item name (name ext), cost, tag1(tag1 info), tag2(tag2 info)
```

```
[Disads]
<category>
disad item name (name ext), cost, tag1(tag1 info), tag2(tag2 info)

[Quirks]
<category>
quirk item name (name ext), cost

[Skills]
<class>
skill item name (name ext), type, tag1(tag1 info), tag2(tag2 info)

[Spells]
<college>
spell item name (name ext), tag1(tag1 info), tag2(tag2 info)

[Packages]
<category>
{Package Item=description}
ADD=AD:predefined advantage item .at. level
CREATE=AD:new advantage item .at. level
MESSAGE=message text to be displayed

[Groups]
<group name>
SK:Skill name

[Mods]
<group name>
item name, modifier cost, tag1(tag1 info), tag2(tag2 info)
```

Section Detail Information

This section will cover the information that you need to write each section of a GCA v2 data file.

Version

The Version section is the smallest and easiest section to do, but it is extremely important that you do it correctly, or the file will be read as if it were a version 1 file, and GCA will merrily read your data into incomprehensible formats internally, making them rather useless to you. Of course, GCA does not modify your data files. The complete Version section looks like this:

```
Version=2
```

The line containing the word ‘Version=’ followed by the version number of the data file format being used, in this case 2, makes up the entire Version section. The Version line must be the absolute first line of the data file, coming before everything else, including comments. Do not put comments before this line!

Author

Tags available: none.

The Author section is also pretty simple: everything following the [Author] tag in the file, until it reaches a new tag, or the end of the file, is considered Author information. All Author information is displayed to the user when they select the About the Data Files... menu option from the Help menu. If you want to include information in comments in the data file that shouldn’t be displayed to the user, put it before or after the Author block.

A sample Author block may look something like this (this one comes from the MAGIC.DAT source file):

```
[AUTHOR]
*****************************************************
* GURPS(R) Character Assistant
* MAGIC.GDF Data File
* Created:  October 20, 1995
* Modified: December 4, 1996
* Requires GCA v2.0 or later!
*
* This data file contains information taken from
* the GURPS Magic source book. GURPS Magic is
* Copyright (c) 1989, 1990 by Steve Jackson Games
* Incorporated. All rights reserved.
*
* Information in this file includes:
*     Advantages, including bonuses
*     Skills, including skill classes, defaults,
*         bonuses, needs
*     Spells, including colleges, needs (prereqs)
*****************************************************
* Contact: Armin D. Sykes
*          72700.1724@compuserve.com
*          ArminSykes@aol.com
*****************************************************
* GURPS is a registered trademark of Steve Jackson
* Games Incorporated, used with GCA by permission
* of Steve Jackson Games. All rights reserved.
*
```

```
* The GURPS game is copyright (c) 1986-1989, 1991-
* 1994 by Steve Jackson Games Incorporated. This
* product includes copyrighted material from the
* GURPS game, which is used by permission of Steve
* Jackson Games Incorporated. All rights reserved
* by  Steve Jackson Games Incorporated.
**************************************************
```

I strongly encourage you to use this or a similar format for your Author block (without the indenting that I use here to set the example apart, of course). If you want to just copy the block out of one of the included data files (not BASIC.DAT, though, since copyright information for that book isn't quite the same as for the others) and replace the appropriate information with that for your own info, that would be great.

Information that should most definitely be included in your Author block include: the name of the data file and the program that it's for, the copyright notice for the source book(s) used to create the data file (*don't forget this!*), what the data file includes, and contact information so people can get in touch with you if they have questions or problems.

Bonus

Tags available: none.

The Bonus section is specifically for giving bonuses that derive from having certain attribute scores or levels. Here's an example of a piece of the Bonus section from the BASIC data file:

```
[BONUS]
<IQ=10>
+1 SK:Beam Weapons
+1 SK:Guns
<IQ=11>
+1 SK:Beam Weapons
+1 SK:Guns
<IQ>=12>
+2 SK:Beam Weapons
+2 SK:Guns
```

Each sub-head enclosed within the angle brackets, like <IQ=10>, tells GCA that the following bonuses apply only when that particular evaluation is true. Within the sub-head, the format is ITEM EVALUATION SCORE, where the valid ITEMs any legal single GCA item (ad, disad, skill, etc.), with applicable prefix header, if any; EVALUATION is something like less than (<), greater than (>), equal to (=), etc.; and SCORE is the score or level for the ITEM used for the evaluation.

Within each subsection, the bonuses are written as +BONUS GIVETO, where +BONUS is the bonus (+ required) given to GIVETO. GIVETO *must* use the special prefix tags (see Prefix Tags) that tell GCA whether the following items are skills, spells, ads, etc. The prefix tag should be followed immediately by the name of the item being given the bonus, exactly as it is used in the appropriate data section elsewhere.

While it is called the Bonus section, you can also apply penalties here, in the same fashion. Simply use -PENALTY (the - is required to tell GCA that it is a penalty) instead of +BONUS. For example, if you like to penalize less agile people when using Jeweler skill, you could do it like this within the Bonus section:

```
<DX<8>
-1 SK:Jeweler
```

Note that if the evaluation for each section specified is true, then each bonus will be applied. This means that you can do cumulative bonuses by listing incremental adjustments to the SCORE required. For example:

```
<DX>10>
+1 SK:Sliding
<DX>11>
+1 SK:Sliding
```

A character with a DX of 10 or less would receive no bonus, a character with an 11 DX would receive the +1 bonus from the first evaluation, and a character with a DX of 12 or higher would receive the bonus from the first and second evaluations, for a total bonus of +2.

Stats

Tags available: basevalue( ), display( ), doubleinplay( ), down( ), maxscore( ), minscore( ), round( ), step( ), symbol(), and up( ). Tags need not be in any particular order.

The Stats section allows you to define new stats/attributes for characters, or redefine attributes that are predefined by GCA.

To define a new stat, or to redefine an existing stat, you must specify the stat name followed by any required tags to complete the item. The first item of any stat item must be the name of the item, and this is the only required part of the item. Following the name, separated by commas, should be a list of the tags required to specify to item completely.

Here is a small sample Stats section that redefines Fatigue and Hit Points, adds a new stat called Smellyness, and changes the costs for the ST stat.

```
[STATS]
ST, step(1), up(5/5), down(-5/-5), doubleinplay(yes)
Fatigue, basevalue(HT), display(no)
Hit Points, basevalue(ST), display(no)
Smellyness, basevalue(20-HT), display(yes)
```

At this time, GCA supports scaled stats (like ST or HT) or figured stats (like Hit Points and Fatigue). However, the only way to adjust any of the stats beyond the basic four (ST, DX, IQ and HT) at this time is to use advantages and disadvantages that give a bonus or penalty to the score. We hope to have an improved Attributes window in the future that will allow you to directly adjust scaled stats other than the basic four.

Predefined Stats

Here is a list of all the stats that are currently built into GCA for your characters: ST, DX, IQ, HT, Will, Fatigue, Hit Points, Speed, Move, Dodge, Vision, Hearing, Taste/Smell, Swim, No Encumbrance, Light Encumbrance, Medium Encumbrance, Heavy Encumbrance, X-Heavy Encumbrance, PD, and DR. As of v2.1 of GCA, these other stats are built in to GCA: Parry, Block, No Encumbrance Move Reduction, Light Encumbrance Move Reduction, Medium Encumbrance Move Reduction, Heavy Encumbrance Move Reduction, X-Heavy Encumbrance Move Reduction, No Encumbrance Move, Light Encumbrance Move, Medium Encumbrance Move, Heavy Encumbrance Move, and X-Heavy Encumbrance Move. These are built in, are always available, and sometimes print in special places on the character sheet. Additional built in stats may be added later.

You must be careful to use the names for stats that GCA expects if you are redefining an attribute, or using one attribute in the calculation for the base value of another attribute. Case is ignored, but spelling and such must be identical. For example, if you want to redefine the X-Heavy Encumbrance stat, you must spell it that way, or you will simply create a new stat.

Note that if you use the symbol() tag for a stat, you may use the name defined in that tag in calculations that require the stat, but you must still use the full name of the stat if you intend to redefine it.

Ads

Tags available: cost( ), gives( ), mods( ), needs( ), notes( ), page( ) and upto( ). Tags need not be in any particular order.

The Ads section contains a list of available advantages, formatted to contain information in a way that GCA can understand.

There are only two required pieces of information for each item in the Ads section: the name of the item and the cost of the item. All parts of this item should be separated from other parts using commas.

The name of the item is very simple: it is the name of the item, using none of the restricted characters. If you enclose a part of the name within parenthesis, that part of the name will be considered the name extension. The name extension allows you to include extra information about the name or the item without affecting how GCA sees the name of the item. You may use commas in the name extension, but the other restricted characters listed above should still not be used. Unless you specifically tell GCA to pay attention to the name extension (such as by referring to it somewhere), it is usually ignored.

The cost of the item is also very simple, but has a couple of different forms. The simplest form is that of a simple number, such as 10. This form is appropriate when the advantage has only a single cost and can not increase in number of levels. The other form of the cost is a split cost, such as 5/5 or 15/10. This form designates that the advantage has more than one possible level and the cost should be handled accordingly. The first number of the split cost is the cost for taking the first level of the advantage. The second number of the split cost is the cost for each additional level of the advantage, after the first. So, the cost for the first level of an advantage with a cost of 15/10 would be 15 points, while the cost for the second level of the advantage would be 25 points, the cost of the third level of the advantage would be 35 points and so on.

The split cost notation for the cost of the item can also be extended to cover items that have multiple levels that do not increase cost in a flat progression. For example, say you have the advantage Schlemping, which costs 5 points for first level, 15 points for second level, 35 points for third level and 65 points for fourth level. Since the cost differences are 5/10/20/30, you could not handle this with a simple split cost. Instead, use an extended split cost, writing the cost of the item just as I did here:

```
5/10/20/30
```

GCA will know that 5 is the cost of the first level and that the costs of the other levels correspond to adding the other costs to the cost of the previous level. This is the same in functionality as a simple split cost, but allows GCA to pay attention to more levels if needed. Please note that, as with simple split costs, the last cost given is used for all levels beyond those explicitly listed, so if only a few levels are allowed, you must remember to use the upto( ) tag to limit the level the advantage can reach.

Note that while the cost of the item should be considered required, you may optionally use the cost( ) tag to specify it. The contents of the cost( ) tag should be specified exactly as listed in the information given above, but using the cost( ) tag allows for the placement of the cost information to fall anywhere in the list of tags for the item. If you do not use the cost( ) tag, the cost information must be the first item specified following the name of the item.

Following the name of the item may be any other items needed for the particular item, using the available tags for this type of item. If the cost( ) tag is not used, cost of the item must be specified before the list of tags.

Sub-headings

You may use sub-headings within this section to break the items into different groups, by enclosing the name of the sub-heading within angle brackets on a line by itself. The sub-headings that you use will be used on the Ad/Disad/Quirk Categories form to allow you to pick your advantages from defined categories instead of from one long list. If an item can fall under one or more sub-headings, you should list it under each sub-heading. It will be treated as a single item when the user selects it, but it will always be listed under all applicable categories.

Here is a small example of an Ads section, taken from the GURPS Basic Set data file:

```
[ADS]
<General>
_Blank Advantage, 0, page(—)
Absolute Direction, 5, gives(+3 SK:Navigation), page(B19)
Absolute Timing, 5, page(B19)
Acute Hearing, 2/2, gives(+1 Hearing), page(B19)
Acute Taste & Smell, 2/2, gives(+1 Taste/Smell), page(B19)
Acute Vision, 2/2, gives(+1 Vision), page(B19)
```

Disads

Tags available: cost( ), gives( ), mods( ), needs( ), notes( ), page( ) and upto( ). Tags need not be in any particular order.

The Disads section contains a list of available disadvantages, formatted to contain information in a way that GCA can understand.

There are only two required pieces of information for each item in the Disads section: the name of the item and the cost of the item. All parts of this item should be separated from other parts using commas.

The name of the item is very simple: it is the name of the item, using none of the restricted characters. If you enclose a part of the name within parenthesis, that part of the name will be considered the name extension. The name extension allows you to include extra information about the name or the item without affecting how GCA sees the name of the item. You may use commas in the name extension, but the other restricted characters listed above should still not be used. Unless you specifically tell GCA to pay attention to the name extension (such as by referring to it somewhere), it is usually ignored.

The cost of the item is also very simple, but has a couple of different forms. The simplest form is that of a simple number, such as -10. This form is appropriate when the disadvantage has only a single cost and can not increase in number of levels. The other form of the cost is a split cost, such as -5/-5 or -15/-10. This form designates that the disadvantage has more than one possible level and the cost should be handled accordingly. The first number of the split cost is the cost for taking the first level of the disadvantage. The second number of the split cost is the cost for each additional level of the disadvantage, after the first. So, the cost for the first level of a disadvantage with a cost of -15/-10 would be -15 points, while the cost for the second level of the disadvantage would be -25 points, the cost of the third level of the advantage would be -35 points and so on.

The split cost notation for the cost of the item can also be extended to cover items that have multiple levels that do not increase cost in a flat progression. For example, say you have the disadvantage Non-schlemping, which costs -5 points for first level, -15 points for second level, -35 points for third level and -65 points for fourth level. Since the cost differences are -5/-10/-20/-30, you could not handle this with a simple split cost. Instead, use an extended split cost, writing the cost of the item just as I did here:

```
-5/-10/-20/-30
```

GCA will know that -5 is the cost of the first level and that the costs of the other levels correspond to adding the other costs to the cost of the previous level. This is the same in functionality as a simple split cost, but allows GCA to pay attention to more levels if needed. Please note that, as with simple split costs, the last cost given is used for all levels beyond those explicitly listed, so if only a few levels are allowed, you must remember to use the upto( ) tag to limit the level the disadvantage can reach.

Note that while the cost of the item should be considered required, you may optionally use the cost( ) tag to specify it. The contents of the cost( ) tag should be specified exactly as listed in the information given above, but using the cost( ) tag allows for the placement of the cost information to fall anywhere in the list of tags for the item. If you do not use the cost( ) tag, the cost information must be the first item specified following the name of the item.

Following the name of the item may be any other items needed for the particular item, using the available tags for this type of item. If the cost( ) tag is not used, cost of the item must be specified before the list of tags.

Sub-headings

You may use sub-headings within this section to break the items into different groups, by enclosing the name of the sub-heading within angle brackets on a line by itself. The sub-headings that you use will be used on the Ad/Disad/ Quirk Categories form to allow you to pick your disadvantages from defined categories instead of from one long list. If an item can fall under one or more sub-headings, you should list it under each sub-heading. It will be treated as a single item when the user selects it, but it will always be listed under all applicable categories.

Here is a small example of a Disads section, taken from the GURPS Basic Set data file:

```
[DISADS]
<Social>
Odious Personal Habit, -5/-5, upto(3), page(B26)
Poverty: Struggling (x1/2), -10, page(B16)
Poverty: Poor (x1/5), -15, page(B16)
Poverty: Dead Broke (x0), -25, page(B16)
Primitive, -5/-5, page(B26)
Reputation, -5/-5, upto(4), mods(Reputation), page(B17)
Social Stigma, -5/-5, upto(4), page(B27)
```

Quirks

Tags available: cost( ), needs( ), notes( ) and page( ). Tags need not be in any particular order.

The Quirks section contains a list of available quirks, formatted to contain information in a way that GCA can understand.

There are only two required pieces of information for each item in the Quirks section: the name of the item and the cost of the item. All parts of this item should be separated from other parts using commas.

The name of the item is very simple: it is the name of the item, using none of the restricted characters. If you enclose a part of the name within parenthesis, that part of the name will be considered the name extension. The name extension allows you to include extra information about the name or the item without affecting how GCA sees the name of the item. You may use commas in the name extension, but the other restricted characters listed above should still not be used. Unless you specifically tell GCA to pay attention to the name extension (such as by referring to it somewhere), it is usually ignored.

The cost of the item is also very simple and is usually just -1. However, you may assign any point value you wish to the quirk, so you can use 0 point items as quirks if you like, or -2 points, whatever.

Note that while the cost of the item should be considered required, you may optionally use the cost( ) tag to specify it. The contents of the cost( ) tag should be specified exactly as listed in the information given above, but using the cost( ) tag allows for the placement of the cost information to fall anywhere in the list of tags for the item. If you do not use the cost( ) tag, the cost information must be the first item specified following the name of the item.

Following the name of the item may be any other items needed for the particular item, using the available tags for this type of item. If the cost( ) tag is not used, cost of the item must be specified before the list of tags.

Sub-headings

You may use sub-headings within this section to break the items into different groups, by enclosing the name of the sub-heading within angle brackets on a line by itself. The sub-headings that you use will be used on the Ad/Disad/ Quirk Categories form to allow you to pick your quirks from defined categories instead of from one long list. If an

item can fall under one or more sub-headings, you should list it under each sub-heading. It will be treated as a single item when the user selects it, but it will always be listed under all applicable categories.

Here is a small example of a Quirks section, taken from the GURPS Basic Set data file:

```
[QUIRKS]
<General>
_Unused Quirk 1, -1, page(B41)
_Unused Quirk 2, -1, page(B41)
_Unused Quirk 3, -1, page(B41)
_Unused Quirk 4, -1, page(B41)
_Unused Quirk 5, -1, page(B41)
Delusion (Quirk), -1, page(B32)
Quirk, -1, page(B41)
Vow (Quirk), -1, page(B37)
Trademark, -1, page(B241)
```

Powers

Tags available: cost( ), gives( ), mods( ), needs( ), notes( ), page( ) and upto( ). Tags need not be in any particular order.

The Powers section of the data file is identical in all ways to the Ads section, except where it ends up in the program and on the character sheet. Refer to the Ads section for a detailed explanation of what can be done in the Powers section.

Here is a small example of a Powers section, taken from the GURPS Psionics data file:

```
[POWERS]
<Psi: Antipsi>
Antipsi, 3/3, mods(Antipsi, Psi Enhancements, Psi Limitations), page(P10)
Antipsi (single skill), 2/2, _
   mods(Antipsi, Psi Enhancements, Psi Limitations), page(P10)

<Psi: Astral Projection>
Astral Projection, 3/3, _
   mods(Astral Projection, Psi Enhancements, Psi Limitations), page(P10)
Astral Projection (Astral Sight only), 1/1, _
   mods(Astral Projection, Psi Enhancements, Psi Limitations), page(P10)
```

Skills

Tags available: default( ), gives( ), mods( ), needs( ), notes( ), page( ), stat( ) and type( ). Tags need not be in any particular order.

The Skills section contains a list of available skills, formatted to contain information in a way that GCA can understand.

There are only two required pieces of information for each item in the Skills section: the name of the item and the type of the item. All parts of this item should be separated from other parts using commas.

The name of the item is very simple: it is the name of the item, using none of the restricted characters. If you enclose a part of the name within parenthesis, that part of the name will be considered the name extension. The name extension allows you to include extra information about the name or the item without affecting how GCA sees the name of the item. You may use commas in the name extension, but the other restricted characters listed above

should still not be used. Unless you specifically tell GCA to pay attention to the name extension (such as by referring to it somewhere), it is usually ignored.

The type of the item is also very simple, as this is usually the type of the skill as specified by GURPS, such as M/H or P/E. When GURPS specifies the type of the skill, that is the information that should be specified here. There are some exceptions to this for special cases, however. All Supers and Psionics skills should use the form S/VH or S/H instead. The S/ types are treated like M/ type skills, but things that affect M/ type skills (such as Eidetic Memory) will not affect the S/ type skills. All skills that are treated as maneuvers, such as the maneuvers from the Martial Arts books, should use the form MA/A or MA/H, as appropriate. This form tells GCA that the item uses the maneuver table instead of the normal skill table.

Note that while the type of the item should be considered required, you may optionally use the type( ) tag to specify it. The contents of the type( ) tag should be specified exactly as listed in the information given above, but using the type( ) tag allows for the placement of the type information to fall anywhere in the list of tags for the item. If you do not use the type( ) tag, the type information must be the first item specified following the name of the item.

Following the name of the item may be any other items needed for the particular item, using the available tags for this type of item. If the type( ) tag is not used, type of the item must be specified before the list of tags.

Sub-headings

You may use sub-headings within this section to break the items into different groups, by enclosing the name of the sub-heading within angle brackets on a line by itself. The sub-headings that you use will be used on the Skill Classes form to allow you to pick your skills from defined classes instead of from one long list. If an item can fall under one or more sub-headings, you should list it under each sub-heading. It will be treated as a single item when the user selects it, but it will always be listed under all applicable classes.

Here is a small example of a Skills section, taken from the GURPS Basic Set data file:

```
[SKILLS]
<Animal>
Animal Handling, M/H, default(IQ-6), page(B46)
Falconry, M/A, default(IQ-5), page(B46)
Packing, M/H, default(IQ-6, SK:Animal Handling-6),_
   needs(SK:Animal Handling), page(B46)

<Artistic>
Artist, M/H, default(IQ-6), page(B47)
Bard, M/A, default(IQ-5, SK:Performance-2), page(B47)
Calligraphy, P/A, default(DX-5, SK:Artist-2),_
   needs(ADS:Literacy), page(B47)

<Athletic>
Acrobatics, P/H, default(DX-6), page(B48)
Bicycling, P/E, default(DX-4, SK:Motorcycle), page(B48)
Breath Control, M/VH, page(B48)
```

Spells

Tags available: castingcost( ), duration( ), needs( ), notes( ), page( ), time( ) and type( ). Tags need not be in any particular order.

The Spells section contains a list of available spells, formatted to contain information in a way that GCA can understand.

There is only one required piece of information for each item in the Spells section: the name of the item. All parts of

this item should be separated from other parts using commas.

The name of the item is very simple: it is the name of the item, using none of the restricted characters. If you enclose a part of the name within parenthesis, that part of the name will be considered the name extension. The name extension allows you to include extra information about the name or the item without affecting how GCA sees the name of the item. You may use commas in the name extension, but the other restricted characters listed above should still not be used. Unless you specifically tell GCA to pay attention to the name extension (such as by referring to it somewhere), it is usually ignored.

Following the name of the item may be any other items needed for the particular item, using the available tags for this type of item.

Sub-headings

You may use sub-headings within this section to break the items into different groups, by enclosing the name of the sub-heading within angle brackets on a line by itself. The sub-headings that you use will be used on the Spell Colleges form to allow you to pick your spells from defined colleges instead of from one long list. If an item can fall under one or more sub-headings, you should list it under each sub-heading. It will be treated as a single item when the user selects it, but it will always be listed under all applicable colleges.

In addition, sub-headings may have codes included in them by separating the code from the main sub-heading name by a colon. You should make sure that the code for each college is unique. The codes are available mainly because they were used in earlier versions of the program and some users like to see the code listed along with the spell in the regular listing of spells on the Spells form.

Here is a small example of a Spells section, taken from the GURPS Basic Set data file:

```
[SPELLS]
<Animal:An>
Beast-Soother, needs(Persuasion | ADS:Animal Empathy), page(B155)
Beast Summoning, needs(Beast-Soother), page(B155)
Reptile Control, needs(Beast-Soother), page(B155)

<Communication & Empathy:CE>
Sense Life, page(B155)
Sense Foes, page(B155)
Sense Emotion, needs(Sense Foes), page(B155)
```

Packages

Tags available: none.

The Packages section allows you to specify lists of items that should be added to the character in one step. Packages are a convenient way to create pre-specified races, careers, or martial arts styles.

The Packages part of a data file is unlike any of the other sections described here, due to the special nature of packages. Packages begin after the [Packages] header, and sub-headings (the names that go on the tabs for the Packages window) under packages are denoted by the < > angle brackets, just like other sections.

However, things vary from that point. The Packages section does not use tags of its own, so it needs to specify the name of each package within a sub-heading by using another form of marker. To do this, the curly braces { } are used.

Within the braces is the name of the individual package, separated from the simple description of the package by an equals = sign. For example, the package Test would be noted like this:

```
{Test=45pts - This is the description of a test package}
```

Note that the cost: part is just part of the description, for the user to see how much it costs, and does not actually mean anything to GCA; the real cost of the package is whatever all the items in the package actually add up to when added to the character.

Under the name of the package will be a list of items using four possible commands: ADD, CREATE, MESSAGE, or EMESSAGE. Each of these commands is separated from the body of the item by an = sign, and each item must be listed on a separate line. The ADD command adds an item from the general Available lists to the character. The CREATE command creates a brand new item that is not necessarily listed in the Available lists. The MESSAGE and EMESSAGE commands display a message to the user.

ADD

The ADD command is very simple, and looks like this:

```
ADD=ADS:Animal Empathy .AT. 1
```

The .AT. part is kind of like the TO part in a gives( ), in that it tells GCA exactly where to break the item apart. The periods beginning and ending the .AT. separator are required. The part of the item specified after the .AT. tells GCA what level you want the item to be that you are ADDing to the character. The remainder of the ADD item is the name of the item that you want to add to the character, as it is listed in the Available lists, prefixed with the appropriate prefix tag. You can ADD another package; the prefix tag for a package is either PA: or PACK:.

If you leave off the .AT. separator, a 1 is usually assumed.

To change one of the four primary attributes, use an ADD type that looks like this (replace ST with the appropriate attribute):

```
ADD=ST .AT. 12
```

Please note that changing an attribute in this way is exactly like changing the attribute on the Attributes window by hand, and is not the way you should change attributes if you are creating a race for GURPS (unless you are using the ‘They cost what they cost’ rule). If you are changing the base score for an attribute for a race, you should use the attribute adjustment method instead, which looks like this (replace ST with the appropriate attribute):

```
ADD=STAdj .AT. +2
```

Alternatively, a method that some people prefer is to use the Create command (described below) to create an advantage that Gives the proper attribute adjustments to the character all at once, with a net cost adjustment for what they should cost.

You can also use the Add method to change the race of the character, like this:

```
ADD=Race .AT. PigBoyz
```

CREATE

The CREATE command is a little more complicated than the ADD command, but that’s only because it has more work to do. The CREATE command looks something like this:

```
CREATE=ADS:Heightened Chi, 10/5, upto(3),_
   needs(ADS:Trained by a Master), gives(+1 Will) .AT. 1
```

As you can see, there is more going on. The bulk of the item, between the = and the .AT., is all the information that GCA needs to create the item. Note that this is listed in exactly the same way as it would be listed in the Ads, Disads, etc., portion of the data file, with one exception: you must include a prefix tag, so GCA will know what kind of item is being created.

Please note that if you use a CREATE to create an item with the exact same name as an item already listed in an Available list, GCA will treat it like an ADD item instead.

The CREATE tag also uses the .AT. separator at the end, so that it can give the character the correct level of the created item after it has been created.

Note: the CREATE tag creates the item and adds it only to the character. The created item does not get added to the general Available lists, so if the user deletes the item, he'll have to recreate it from scratch, or add the package again.

You can not CREATE another package. If you leave off the .AT. part, a 1 is usually assumed.

MESSAGE

The Message command is the simplest of the bunch, it looks like this:

```
MESSAGE=This is a sample message.
```

Everything after the = sign is displayed to the user in a message box when they add the package to their character. The entire text of the message must follow the = sign on the same line, although you can use the line continuation character (_) to break the message into multiple lines in the data file.

EMESSAGE

The EMESSAGE command is similar to the MESSAGE command, but uses a somewhat expanded form to provide information to the user. It looks like this:

```
EMESSAGE=This is the prompt for the message_
   .WITH. This is the first line of the message~P_
   and this is the second line of the message.
```

Everything after the = sign is displayed to the user in a message box when they add the package to their character. The difference is that the part before the .WITH. separator is displayed as a caption or a prompt, and the part after the .WITH. separator is displayed within the message area of the window.

The ~P marker means that GCA should include a line break at that point in the text. The ~P can be used in either the prompt or the message text, and can also be used in a standard MESSAGE command item.

Modifiers for ADD and CREATE items

You can also add modifiers to an item being ADDed to a character using the .WITH. and .AND. keywords (as long as that item type supports mods). Use .WITH. to specify that you are going to be adding some modifiers, and use .AND. to separate modifiers if adding more than one.

When adding modifiers, you are basically creating them on the spot, so you must specify all items that must be included, including the name, cost, any gives( ), etc. GCA will never look up mods created in an ADD or CREATE command in the general list of modifiers to find missing info. For example,

```
CREATE=DI:Dependency (sentient humanoid blood), 0_
   .with. Infrequent, -20_
   .and. Monthly, x1
```

creates the disad Dependency (sentient humanoid blood) at a cost of 0, then uses the mods that are included in the CREATE line to set the final cost. If you left off the cost or value adjustments in the mod items, GCA would not find them elsewhere, even if Infrequent and Monthly were listed in a Mods group somewhere.

Groups

Tags available: none.

The Groups section allows you to group names of items together so that they can be considered together for satisfaction of a Need or application of a bonus or penalty. Within the Groups section, sub-headings within angle brackets denote the names of different groups, which can then be used in Needs or Gives tags by using the GR: prefix tag.

Using Groups allows you to apply the same bonus to multiple items in a convenient fashion, such as adding the bonus from the Voice advantage to a bunch of different skills by saying gives(+2 GR:Voice) instead of listing every item right there. Also, by using a Group, you can easily add items to the list of items in the group and all items that need or affect that group will automatically affect the additional items as well.

You can also use Groups to satisfy the Needs of an item when the Needs may not be clearly defined in the text, such as a spell that needs two Seek spells. There are a variety of Seek spells, but they are parts of other colleges, not one overall Seek college, so using a Group would allow for an easy definition of what a Seek spell is and therefore allow GCA to satisfy that Need.

Here is an example of a Groups section, taken from the GURPS Basic Set data file:

```
[GROUPS]
<Math>
SK:Mathematics

<Computer>
SK:Computer Programming

<Engineering>
SK:Engineer

<Music>
SK:Singing
SK:Musical Instrument

<Voice>
SK:Bard
SK:Diplomacy
SK:Performance
SK:Politics
SK:Savoir-Faire
SK:Sex Appeal
SK:Singing

<Seek Spells>
SP:Seek Earth
SP:Seek Water
```

Mods

Tags available: none.

The Mods section contains modifiers that can be applied to advantages, disadvantages, powers and skills. These modifiers are very much like special kinds of advantages or disadvantages and share many of the same kinds of items, although there are also many differences.

Within the Mods section, sub-headings surrounded by angle brackets denote specific, separate groups of modifiers. Each group can only be applied to ads/disads or skills that specifically name that group within their Mods tags.

There are two required pieces of information for each item in the Mods section: the name of the item and the value of the item. All parts of this item should be separated from other parts using commas.

The name of the item is very simple: it is the name of the item, using none of the restricted characters. Mods does not recognize name extensions, so if you do use parenthesis, the text within will be considered part of the name of the mod.

The value of the item is also very simple, but has a couple of different forms. The simplest form is that of a simple positive or negative number, such as +10 or -5. This form is appropriate when the modifier has only a single base value and can not increase in number of levels. Another form of the value is a split cost, such as +5/+5 or +15/+10. This form designates that the Mod has more than one possible level (called stacking) and the cost should be handled accordingly. The first number of the split value is the value for taking the first level of the Mod. The second number of the split value is the value for each additional level of the modifier, after the first. So, the value for the first level of a modifier with a value of +5/+5 would be +5 points, while the value for the second level of the modifier would be +10 points, the value of the third level would be +15 points and so on.

All of the values given in the previous paragraph are called addition modifiers, because they add together, even if their value is negative. There are also two other kinds of modifiers: multiplication and percentage. Where the addition mods add to the value of the modified item, multiplication mods multiply the value of the item by some number, possibly a fraction. The value of a multiplier mod would be written as times some number, such as *5 or *1/2. You can not use stacking on multiplication mods, because GCA will always assume that it is a fractional multiplier.

Percentage mods add or subtract some percentage of the item’s current value back to the item. Percentage mods would be written as a positive or negative percentage value, such as +50% or -30%; in this way you can think of them as special kinds of addition mods. You may use stacking with percentage mods. Where addition and multiplication mods use a set value to affect the final cost of the modified item, percentage mods use of percentage values means that the cost of the value is affected by the current cost of the item, before the mod is applied. For example, an item with a current cost of 20 and a modifier of +50% would add 50% of the cost of the item back to the cost, for a resultant cost of 30.

Please note: all addition mods are done first, so that they may be used to create, or help determine, the base cost of the item being modified. The multiplier mods are done second, with any rounding done accordingly. The percentage mods are done last, after all percentage mods are added together to get a final percentage value that should be applied to the cost of the item.

Following the name and the cost of the item may be any other items needed for the particular item, using the available tags for this type of item. Tags that may be used for Mods are: round( ) and upto( ). Tags need not be in any particular order.

Here is a small example of a Mods section, taken from the GURPS Basic Set data file:

```
[MODS]
<Reputation>
People: Everyone, *1
People: Large class, *1/2, round(down)
People: Small class, *1/3, round(down)
Frequency: Always, *1
Frequency: Sometimes (10 or less), *1/2, round(down)
Frequency: Occasionally (7 or less), *1/3, round(down)

<Patron>
Size: Powerful Individual, +10
Size: Group, +10
```

```
Size: Extremely Powerful Individual, +15
Size: Reasonably Powerful Group, +15
Size: Very Powerful Group, +25
Size: National Government, +30
Size: Giant Multi-national Group, +30

<Enemy>
Size: Single above avg. individual, -5
Size: Single formidable individual, -10
Size: Group(3 to 5 normals), -10
Size: Medium Group(6 to 20), -20
Size: Large Group(20 to 1000), -30
Size: Formidable Group(Gov’ts or guilds), -40

<Frequency>
Frequency: Almost all the time (15 or less), *3
Frequency: Quite often (12 or less), *2
Frequency: Fairly often (9 or less), *1
Frequency: Rarely (6 or less), *1/2, round(up)
```

Tags Detail Information

basevalue( )

Applies to: stats.

This tag allows you to specify what should be used for the base value of the item, and whether it should be a constant or a calculated value. To use a constant value, simply specify that value within the parenthesis, like this:

```
basevalue(10)
```

To use math, simply use a math expression within the parenthesis, but be aware of the use of any math characters within the names of items used in the calculation. These are valid basevalue tags using math:

```
basevalue(ST/2)
basevalue((ST+HT)/2)
basevalue(20-"Taste/Smell")
```

Note the use of the double quotes around Taste/Smell to ensure that GCA recognizes it as an item name, not as one item divided by another.

castingcost( )

Applies to: spells.

This tag allows information to be passed to the user about the casting cost of a spell. The information contained within the parenthesis can be of almost any kind, because it will be ignored by the program.

cost( )

Applies to: advantages, disadvantages, quirks and powers.

This tag allows information covering the cost of an item to be included in a location other than immediately following the name of the item. Use of this tag is optional, but if this tag is not used, the cost for the item must be the first item listed after the name of the item. (See the Section Detail information on the particular item for more details.)

default( )

Applies to: skills.

This tag allows skills in the data file to specify from what items they may default and at what penalty, if any. Generally, the defaults should be specified in the same fashion as used by the GURPS books. For example, if the skill can default from DX at -6, the Default tag should be written like this:

```
default(DX-6)
```

If the skill can default from another skill, then it should also be written as specified in the GURPS books, except that the appropriate prefix tag should be used. For example, if the skill can default from Blacksmith skill at -3, the Default tag should be written like this:

```
default(SK:Blacksmith-3)
```

If the skill can default from more than one possible item, they can all be included in the Default tag by separating the different items with commas. For example, if the skill can default from either DX at -6 or Blacksmith skill at -3,

the Default tag should be written like this:

```
default(DX-6, SK:Blacksmith-3)
```

GCA will know that only the best default possible should be used for the character.

Simple math may be used in the default tag items. The purpose for allowing simple math is so that certain items which may default from calculated items can still be used. For example, if an item can default off the Karate parry, which is 2/3 of the Karate skill, the only way to get at that number is to figure it in the Default tag. For example, the Hand-Clap Parry maneuver (from GURPS Martial Arts) Default tag would look like this:

```
default(SK:Judo*2/3-5, SK:Karate*2/3-5)
```

display( )

Applies to: stats.

This tag allows you to specify whether a defined or redefined stat should be listed under the Statistics section of the character sheet or not. Using display(yes) will cause the item to be listed under Statistics; using display(no) will prevent it from being listing listed there.

The default value for this tag is yes.

doubleinplay( )

Applies to: stats.

This tag allows you to specify whether the stat should cost double when the Character In Play check box is checked. Use doubleinplay(yes) when cost should be doubled, and doubleinplay(no) when cost should not be doubled.

This tag only has meaning when applied to a scaleable stat (such as ST or DX) that can be adjusted directly by the user on the Attributes form (which is currently limited to the basic four stats: ST, DX, IQ and HT).

The default value for this tag is yes.

down( )

Applies to: stats.

This tag allows you to specify the costs for a stat when the user lowers the score of the stat below the base value.

This tag only has meaning when applied to a scaleable stat (such as ST or DX) that can be adjusted directly by the user on the Attributes form (which is currently limited to the basic four stats: ST, DX, IQ and HT).

The costs that you specify should be listed in a cost/cost/cost format, where each cost listed is the difference between the current level and the next level of the stat. In the case of the down( ) tag, this would mean that the first cost would be the difference between BASE and BASE-1, the next between BASE-1 and BASE-2, and so on. The last cost specified will determine the cost used for all additional decrements of the attribute score. For example,

```
down(-10/-5/-5/-10)
```

specifies the standard costs for lowering an attribute such as DX from the base value of 10 (or racial average). Lowering from 10 to 9 costs -10, from 9 to 8 costs an additional -5, or -15 total, from 8 to 7 costs another -5, or -20 total, and so on. Since -10 is the last cost specified, -10 will be the cost for each additional decrement of the attribute.

By default, if the Step (see step( ) below) has been specified as anything other than 0, the down( ) list will be the

same as that used for attributes.

duration( )

Applies to: spells.

This tag allows information to be passed to the user about the duration of a spell. The information contained within the parenthesis can be of almost any kind, because it will be ignored by the program.

gives( )

Applies to: advantages, disadvantages, powers, skills and mods.

This tag allows items in the data file to specify what bonuses or penalties should be applied to other items on the character if they take the current item. Generally, the items given to the character should be specified in the same fashion as used by the GURPS books. For example, if the item grants a +1/8 bonus to Speed for each level, the Gives tag should be written like this:

```
gives(+1/8 Speed)
```

If the item gives to a skill instead of an attribute, the appropriate prefix tag should be used. For example, if the item gives a +3 to the Navigation skill, the Gives tag should be written like this:

```
gives(+3 SK:Navigation)
```

If the item can be applied to all the items in a Group specified elsewhere in the data file, the name of the Group and the appropriate prefix tag should be used to apply the bonus to all items belonging to that group. For example, if the item gives a +2 bonus to everything in the Group Voice, the Gives tag should be written like this:

```
gives(+2 GR:Voice)
```

If the item has more than one item to which it applies bonuses or penalties, they can all be included in the Gives tag by separating the different items with commas. For example, if the item gives a +3 bonus to Climbing skill and to Mechanic skill, the Gives tag should be written like this:

```
gives(+3 SK:Climbing, +3 SK:Mechanic)
```

The items given by the Gives tag do not have to be only bonuses. For example, if the item gives a -1 penalty to the Merchant skill, the Gives tag should be written like this:

```
gives(-1 SK:Merchant)
```

Also, the items given by the Gives tag do not have to be only pluses or minuses. In certain special cases, the given item can by a multiplier. In this case, an ‘x’ or an ‘*’ should be used to denote multiplication. For example, if the item gives double the points spent on Mental skills, the Gives tag should be written like this:

```
gives(*2 MentalSkillPoints)
```

If the item also gave a +1 bonus to all spells the character takes, then the Gives tag should be written like this:

```
gives(*2 MentalSkillPoints, +1 Spells)
```

Bonuses or penalties applied from the Gives tag are automatically increased by GCA to match the appropriate level taken of the item, so all items given in the tag should be specified as if they were per level adjustments. For example, if two levels of the item from the previous example were taken, the resultant bonus to the character would be *4 to all points spent on Mental skills and +2 to all spells.

If you need a bonus that is applied to an item only one time, just for having the bonus granting item, that does not increment according to the levels taken of the item, you must use the '=' sign at the beginning of the bonus instead. For example:

```
gives(=6 SK:Accounting)
```

This will add six levels to the Accounting skill, but only one time, so taking additional levels in the bonus-granting item will not increase the bonus the Accounting skill will receive from it. Note that only addition bonuses of this kind are supported and that using '=+' is the same as using '=' at the beginning of the section.

Giving Points

Gives( ) can be used to give points to skills. The structure of the tag is the same, except the end of the bonus part must end with 'pts' and there can be no spaces between the 'pts' part and the bonus. For example:

```
gives(+6pts SK:Accounting)
```

This bonus grants 6 points to the Accounting skill per level of the bonus granting item. Note that you can also give non-incrementing points, such as:

```
gives(=6pts SK:Accounting)
```

which gives a one time bonus of 6 points to the Accounting skill, regardless of the level of the item giving the bonus.

Here are some items that can be used in a Gives tag:

ST denotes that the ST score should be affected.

DX denotes that the DX score should be affected.

IQ denotes that the IQ score should be affected.

HT denotes that the HT score should be affected.

Willdenotes that the Will score should be affected.

Fatigue denotes that the Fatigue score should be affected.

Hit Points denotes that the Hit Points score should be affected.

MentalSkillPoints denotes that all skill points for Mental skills should be affected. This will only affect skills of type M/something, so P, MA and S skills all will be unaffected.

Spells denotes that all spells taken by the character should be affected.

Skills denotes that all skills taken by the character should be affected.

Expanded Format

The Gives tag also has an expanded format that can be more readable for some users and is required for certain expanded features of the tag. Basically, the expanded format makes use of keywords to help GCA determine where different things are in the data file. The most common keyword is TO and it is used to separate the bonus from the item being given the bonus. For example, a gives using TO might look like this:

```
gives(+3 to SK:Climbing, +3 to SK:Mechanic)
```

Note that there must be a space to either side of the keyword for GCA to be able to identify it properly. The TO keyword is optional for most cases of Gives, but it is required when using other keywords, or when doing math in

the bonus portion of a Gives item.

The only other keyword currently defined in Gives is UPTO and it is used to limit the amount of a bonus that can be received from an item. For example:

```
gives(+1/8 to Speed upto 2)
```

would limit the bonus to a maximum of +2, regardless of how many levels of the item were actually taken.

maxscore( )

Applies to: stats.

This tag tells GCA the maximum score allowed for a scaled attribute. Currently ignored for calculated attributes.

The default value for this tag is 1000.

minscore( )

Applies to: stats.

This tag tells GCA the minimum score allowed for a scaled attribute. Currently ignored for figured attributes.

The default value for this tag is 1.

mods( )

Applies to: advantages, disadvantages, powers and skills.

This tag allows items to specify what modifiers from the available Mods groups can be applied to the current item. If one or more Mods have been specified in this tag, the Mods button will be active in the Edit form, allowing the user to activate the Mods form and select modifiers to use on the current item.

For example, the Ally advantage can be modified by two sets of modifiers, those that set the base cost of the advantage and those that modify the cost by the frequency of appearance of the ally. This is how the Mods tag for Ally would look:

```
mods(Ally, Frequency)
```

needs( )

Applies to: advantages, disadvantages, quirks, powers, skills and spells.

This tag allows items to specify any needed requirements or prerequisites for the current item. Generally, all needs should be prefixed with the appropriate prefix tag for the type of need, such as AD: for needed advantages and SK: for needed skills. It is not necessary to use a prefix tag for needed spells within the Spells section of the data file, but it is suggested that you do so, since this will speed up processing.

How these needs are written may vary slightly from the way they are generally written in the GURPS books.

If a particular attribute score is required, the tag should be written as the name of the attribute equal to the minimum value required. For example, if an IQ of 15 is required, the Needs tag should be written like this:

```
needs(IQ=15)
```

GCA knows that any value higher than this will also satisfy the need.

If a particular advantage is required, the tag should be written as the name of the advantage, including the appropri-

ate prefix tag. For example, if Magery is required, the Needs tag should be written like this:

```
needs(ADS:Magery)
```

If the advantage is needed at a particular level, the tag should be written as the name of the advantage equal to the minimum level required. For example, if Magery 3 is required, the Needs tag should be written like this:

```
needs(ADS:Magery=3)
```

GCA knows that any value higher than this will also satisfy the need.

If a particular skill is required, the tag should be written as the name of the skill, including the appropriate prefix tag. For example, if Physics is required, the Needs tag should be written like this:

```
needs(SK:Physics)
```

If the skill is needed at a particular level, the tag should be written as the name of the skill equal to the minimum level required. For example, if a Physics skill of 15 is required, the Needs tag should be written like this:

```
needs(SK:Physics=15)
```

GCA knows that any value higher than this will also satisfy the need.

If a particular spell is required, the tag should be written in the exact same fashion as if it was a required skill, with the exception that the SP: prefix tag should be used instead of the SK: prefix tag.

Please note that the default level required for a needed spell is 12, while for a skill it is only 1.

If one or more items from a Group is required, the tag should be written as the number of items needed from the Group, followed by the name of the Group with the appropriate prefix tag. For example, if 2 spells are needed from the Group Seek Spells, the Needs tag should be written like this:

```
needs(2 GR:Seek Spells)
```

If all of the items from a particular Group are needed, simply omit any number before the name of the Group and the entire group will be considered to be required, like this:

```
needs(GR:Seek Spells)
```

Spell Colleges can be considered special kinds of Groups and you can use the same format as that for Groups to require items from a College. You should not, however, use any prefix tag when referring to a needed College. For example, if you need 4 spells from the Air College, the Needs tag should be written like this:

```
needs(4 Air)
```

This OR That

In many cases, Needs are given as two or more possibilities that may satisfy the Needs for that particular item. The way GCA handles it is to use the OR sign (|). The OR sign (also know as the pipe symbol) can be found over the backslash (\ ) character on most keyboards.

When GCA sees an OR sign in the Needs tag information, it breaks the need into groups based on the OR sign, so everything that is needed to satisfy the Need must be specified on each side of the separator. For example, the Underwater Demolition skill has a prerequisite of the Scuba skill and either the Demolition skill or the Engineer skill. So, what this means is that the Needs for this skill would be satisfied if the character had the Scuba skill and the Demolition skill, or the Scuba skill and the Engineer skill. The Needs tag for the Underwater Demolition skill would look like this:

```
needs(SK:Scuba, SK:Demolition | SK:Scuba, SK:Engineer)
```

Remember, GCA breaks the Needs information apart at the OR, so anything that is always needed must be listed on both sides.

Here is another example, the Summon Elemental spell, in this case for the Earth college. This spell always needs Magery of at least level 1, but then is satisfied if the character has 8 other Earth spells, or 4 other Earth spells and either Summon Air Elemental, Summon Water Elemental or Summon Fire Elemental. Now, knowing this, and that the one item that is required in all cases (Magery 1) must be repeated in each group, the Summon Earth Elemental spell would be recorded like this in the Spells section (the _ characters are to make it easier to read and are only required if you actually break the spell into multiple lines in the data file):

```
Summon Earth Elemental, needs_
(ADS:Magery=1, 8 Earth _
| ADS:Magery=1, 4 Earth, Summon Air Elemental _
| ADS:Magery=1, 4 Earth, Summon Water Elemental _
| ADS:Magery=1, 4 Earth, Summon Fire Elemental _
), page(B156)
```

v2.1 Changes

As of v2.1, Needs can now support additional evaluations, such as >, <, >=, <=, or ==. Those should be pretty obvious as to what they require, with the possible exception of ==. Using == means that the needed item must be exactly equal to the value specified, not equal to or greater than as the = sign means. Note also that >= is the same as = to GCA.

GCA v2.1 also adds math support for the right side of the Needs evaluation, so you could specify a needs like this:

```
needs(SK:Moping=(IQ+HT)/2)
```

notes( )

Applies to: advantages, disadvantages, quirks, powers, skills and spells.

This tag allows information to be passed to the user that will not affect operation of the program. The information contained within the parenthesis can be of almost any kind, because it will be ignored by the program.

page( )

Applies to: advantages, disadvantages, quirks, powers, skills and spells.

This tag allows the item to contain a page reference, so that the user can look up in the appropriate book any clarification that may be needed. Multiple page references can be included by separating them with commas.

You should specify pages using the GURPS standard of book abbreviation plus page within that book, such as B14 for Basic Set, page 14, or FF28 for Fantasy Folk, page 28.

round( )

Applies to: mods and stats.

Mods

This tag is used to tell the program to round up or down, depending on the needs of the Mod it is associated with. If

more than one Mod is used on an item, any Mod that should be rounded up will result in the final modified cost for the item to be rounded up.

If an item should be rounded up, the tag should look like this:

```
round(up)
```

By default, rounding is down, so you shouldn't need to use a Round tag to denote this; however, if you prefer, you can denote rounding down with a tag like this:

```
round(down)
```

Stats

Using the Round tag with stats is similar to using it with Mods, but it has a slightly different meaning: what is rounded is the stat value itself, not the final result of some math resulting from using the Stat in a calculation.

If you specify round(up), any value of .5 or higher will be rounded to the next whole number. If you specify round(down), any fractions will be dropped. If you specify round(no), no rounding will be performed.

stat( )

Applies to: skills.

This tag allows the item to specify the Attribute (statistic) associated with the item. This is required for those items where the type of skill does not match the stat used to calculate levels. For example, Running is a P/H skill, but is based on HT instead of the standard DX. In this case, the Stat tag should be written like this:

```
stat(HT)
```

The Running skill would therefore look like this in the data file:

```
Running, P/H, stat(HT), gives(+1/8 Move, -1/8 Dodge), page(B48)
```

(The -1/8 Dodge is because Dodge is based on Move, but officially, Dodge is not affected by the Running bonus).

step( )

Applies to: stats.

This tag allows you to specify the value to be used to increase or decrease a scaled stat when it is incremented one point up or down.

This tag only has meaning when applied to a scaleable stat (such as ST or DX) that can be adjusted directly by the user on the Attributes form (which is currently limited to the basic four stats: ST, DX, IQ and HT).

For example, specifying a Step of 2 for ST would cause the ST score to be incremented by 2 each time you pressed on the Up arrow next to the ST score, but the cost would still increment one step at a time; so, the first click would raise ST from 10 to 12, but the cost would be 10, the next click would raise ST from 12 to 14, but the cost would only be 20, and so on.

symbol()

Applies to: stats.

This tag allows you to specify an alternate name for a stat that can be used in math enabled sections of GCA. Specifying a symbol allows you to use a smaller, simpler name for a stat while still showing the more descriptive

name to the user. For example, if you have a stat named Pheromonic Dissonance, you could specfiy a symbol such as Stink to make your math based on that stat easier, since you'd only have to type Stink into the math expression, instead of the longer Pheromonic Dissonance.

time( )

Applies to: spells.

This tag allows information to be passed to the user about the casting time of a spell. The information contained within the parenthesis can be of almost any kind, because it will be ignored by the program.

type( )

Applies to: skills and spells.

This tag allows the item to specify the type of spell or skill that it is (M/H, P/H, etc).

As of 2.1, this tag no longer restricts spells to M/H or M/VH. You can now specify P/H or M/E and such for spells. Note that spells are still required to be based on IQ, since spells do not support the stat() tag. This tag is still optional if the spell is M/H.

For skills, this tag is optional if the type information is specified immediately after the skill name (see the Section Detail information for Skills), but is required if the type information is specified elsewhere in the item information.

up( )

Applies to: stats.

This tag allows you to specify the costs for a stat when the user raises the score of the stat above the base value.

This tag only has meaning when applied to a scaleable stat (such as ST or DX) that can be adjusted directly by the user on the Attributes form (which is currently limited to the basic four stats: ST, DX, IQ and HT).

The costs that you specify should be listed in a cost/cost/cost format, where each cost listed is the difference between the current level and the next level of the stat. In the case of the up( ) tag, this would mean that the first cost would be the difference between BASE and BASE+1, the next between BASE+1 and BASE+2, and so on. The last cost specified will determine the cost used for all additional increments of the attribute score. For example,

```
up(10/10/10/15/15/20/20/25)
```

specifies the standard costs for raising an attribute such as DX from the base value of 10 (or racial average). Raising from 10 to 11 costs 10, from 11 to 12 costs an additional 10, or 20 total, from 12 to 13 costs another 10, or 30 total, and so on. Since 25 is the last cost specified, 25 will be the cost for each additional increment of the attribute.

By default, if the Step (see step( ) below) has been specified as anything other than 0, the up( ) list will be the same as that used for attributes.

upto( )

Applies to: advantages, disadvantages, powers, skills and mods.

This tag allows the item to specify the number of levels or points that an item may have, if the cost of the item is specified as one that may allow multiple levels. For example, Magery may have up to three levels, so the Magery advantage would be recorded like this:

```
Magery, 15/10, upto(3), gives(+1 Spells), page(B21)
```

The 15/10 cost specifies that the first level costs 15 points and additional levels cost 10 points each. Since only 3 levels are available, it is necessary to use the Upto tag to specify that the item can only go up to three levels.

The Upto tag as shown above lists the number of levels that may be limited by the Upto tag. You may also specify the maximum number of points that can be used in the item, by ending the expression within the tag parenthesis with the abbreviation for points, pts. For example, you could list the Upto tag for the Magery example given above like this instead:

```
upto(35pts)
```

You can also use math in an Upto tag section. You should list any math items in the same way that you would normally write math in a document, using standard order of operations and allowing parenthesis (see Math above).

Just for the sake of example, lets use an Upto tag with some unnecessary extra math in it. For this example, we'll use a made up advantage called Targeting Ears, which is limited in the maximum level allowed by the level of Acute Hearing the character has. Here is a possible Upto tag for our Targeting Ears:

```
upto((3 + 2) * 2 + ADS:Acute Hearing * 5pts)
```

Note that parenthesis are allowed (make sure the (and) pairs balance, with the same number of each!) and a reference is made to the Acute Hearing advantage. This particular advantage is limited to a number of points that can be spent, as denoted by the fact that it ends with pts. Remember your order of operations! Multiplication and division is done before addition and subtraction, so the Acute Hearing level will be multiplied by 5 before being added to the other stuff.

If you wanted to reference the value for the points spent on the Acute Hearing advantage, instead of the level of that advantage as shown in the above example, you should end the name of the item with a space and a pts designation. The space is very important, otherwise GCA will think that the pts designation is part of the advantage name. The above example, using the points of the Acute Hearing advantage instead, would look like this:

```
upto((3 + 2) * 2 + ADS:Acute Hearing pts * 5pts)
```

Skills

While the examples in this section pertain mainly to advantages, Upto applies equally well to skills, where limiting often comes into play with maneuvers.

Skills have two additional references that are allowed: default and prereq. If a skill (including a maneuver) is defaulted from something else, the word default can be used to provide a reference to the item from which the skill is defaulted, without having to know exactly which item that may be. If a skill is limited by the level of its prerequisite item, then the word prereq can be used to provide a reference to the highest value of all possible prereqs that exist for the skill.

For example, if a skill requires either Karate or Judo, but can not exceed the level of its prerequisite skill, the Upto for that maneuver could be written like this:

```
upto(prereq)
```

Of course, math is also supported with this pair of references.

Modifiers

As of v2.1, Modifiers are now allowed to use math in an Upto statement. You still may not specify a number of points, however.

Additional Assistance

If you need additional assistance, try posting your question to the GCA mailing list. To subscribe to the mailing list, send an email message to

```
gca-l-request@lists.sidhe.org
```

with the word SUBSCRIBE as the only text in the body of the message.

The address to send your questions or responses to is

```
gca-l@lists.sidhe.org
```

The GCA-L mailing list is an excellent resource, because it is monitored by the author of GCA and a variety of knowledgeable users, all willing to help.