GCA v2 Mini-Spec

Based on the 31 March 1997 version of the full specification.

Contents

Purpose 3
Restricted Characters 3
Prefix Tags 3
Math 4
Functions 4
@MAX( ) and @MIN( ) 4
@IF( ) 5
Tags Detail Information 5
default( ) 6
gives( ) 6
Giving Points 8
Expanded Format 9
needs( ) 9
This OR That 11
v2.1 Changes 12
upto( ) 12
Skills 13
Modifiers 13
Additional Assistance 13

Purpose

This mini-version of the much larger GCA v2 Data File Format Specification is intended to provide only the information that a user of GCA needs to make better use of certain important tags within GCA. Tags are those parts of items that hold information that GCA uses (the headings on the columns of the Edit window), so gives( ) is a tag, while the information within the parenthesis is the information GCA uses for the gives( ) tag (the information within the columns of the Edit window).

The information contained in this file is taken directly from the full GCA v2 Data File Format Specification (available on the GCA web site: http://www.teleport.com/~armin/gca/), and you can look there for more complete information on the structure GCA uses for data files.

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.

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.

Tags Detail Information

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)
```

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.

Will denotes 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.

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 appropriate 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)
```

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.