EGEE

R-GMA Gin

Document identifier:	**EGEE-JRA1-TEC-gin-manual**
Date:	**May 11, 2006**
Activity:	**JRA1: Middleware Engineering and Integration (UK Cluster)**
Document status:	**DRAFT**
Document link:	

Abstract: This document describes the R-GMA Gin tool

1 INTRODUCTION

Gin is a service to extract information in LDIF format and republish it to R-GMA. It uses a configuration file which contains the mapping from LDIF attributes to RGMA tables and columns.

2 USAGE

To start the Gin service, use:

```
/etc/init.d/rgma-gin (start|stop|status)
```

To run Gin directly from the command line, use:

```
$RGMA_HOME/bin/rgma-gin
```

3 CONFIGURATION

The Gin configuration file is

```
$RGMA_HOME/etc/rgma-gin/gin.conf
```

3.1 AUTOMATIC CONFIGURATION

To automatically configure Gin to publish information from predefined information providers, run:

```
$RGMA_HOME/bin/rgma-gin-config --glite-ce=yes|no --gip=yes|no --fmon=yes|no
```

This will overwrite your previous configuration file. The glite-ce information provider publishes the CE information suitable for a Glite deployment. The gip information provider publishes the CE information for an LCG deployment. Note that you cannot use both of these information providers together (The glite-ce information is essentially a subset of gip).

3.2 MANUAL CONFIGURATION:

To manually configure Gin, edit the file:

```
$RGMA_HOME/etc/rgma-gin/gin.conf
```

The format of this configuration file is described in section 5

4 LOGGING

The Gin log file is:

```
/var/log/glite/rgma-gin.log
```

Logging is controlled by the log4j properties file:

```
$RGMA_HOME/etc/rgma-gin/log4j.properties
```

5 DETAILED CONFIGURATION FILE REFERENCE

The Gin configuration file has the general format:

```
infoprovider <name of info provider>

  table <table>[=<objectclass>]
    column <type> <column>[=<attribute>]
    ...
    [more columns]
    ...
  endtable
  ...
  [more tables]
  ...
  [more options]
endinfoprovider
...
[more info providers]
...
```

Comments are allowed, using the # character. All keywords are case insensitive.

5.1 TABLE DEFINITION

A table definition has the following form.

```
table <table name>[=[<object class>]]
```

Declares a table to be published. The LDIF output will be scanned for a matching object class to identify data which should go into the table. `<table name>` gives the name of the R-GMA table which the data should be published to. If `<object class>` is specified, the LDIF will be scanned for an object class with this name. Otherwise the LDIF will be scanned for an object class with the name `<table name>`.

Example:

```
table GlueSA
```

will search for `GlueSA` in the LDIF file, using data in objects which match that for populating the `GlueSA` table.

```
table GLueSA=GlueSAPolicy
```

will search for objects containing `GlueSAPolicy` and use data in objects which contain that for populating the `GlueSA` table.

5.2 COLUMN DEFINITION

Between a `table` and and `endtable`, there should be a list of columns in the table

```
column:<type> <column>[=[<attribute name>]]
```

This specifies the columns in the table which should be published.

`<type>` defines the SQL type of the column, e.g. `INTEGER` , `VARCHAR(20)` . The attribute value will be interpreted according to the type specified and the information will be rejected if it does not match (e.g. if the data `fred` is specified for an `INTEGER` column).

One exception to the type checking rule occurs for character data if the `truncate` keyword is specified (see section **??**). In this case, `CHAR` or `VARCHAR` data will be truncated to the specified size (e.g. if the data `fred` is specified for a `VARCHAR(2)` column, the value 'fr' will be inserted).

The attribute corresponding to the column is taken from `<attribute name>`, if it is specified. If it is not, then the object class in the table specification is used with the column name appended to it. If no object class was specified for the table, the R-GMA table name with the column name appended to it is used.

Example:

```
table fred=joe
  column:INTEGER mycol
endtable
```

Will populate column `mycol` in table `fred` with the `joemycol` attribute from object containing object class `joe`. The attribute data will be interpreted as an integer.

```
table fred
 column:VARCHAR(25) mycol
endtable
```

Will populate `mycol` in table `fred` with `fredmycol` attribute from object containing object class `fred`. The attribute data will be interpreted as a string of length 25.

```
table fred
  column:REAL mycol=thatAttribute
endtable
```

Will populate `mycol` in table `fred` with `thatAttribute` attribute from object containing object class `fred`. The attribute data will be interpreted as a floating point number.

There is a variation on the syntax for columns. You can specify

```
table fred
  column:VARCHAR(255) mycol=Aye+Bee
endtable
```

It will then construct the value of `mycol` by appending the value of the attribute `Bee` to the value of the attribute `Aye`. Any number of attributes may be strung together in this way. The full name of the attribute must be given (in the same way as `column mycol=Aye` specifies `Aye` as the full name of the attribute).

Sometimes you might want to seperate `Aye` and `Bee` with a specific character, so that they do not just run into each other:

```
table fred
  column:VARCHAR(255) mycol=Aye+/+Bee+:+Sea
endtable
```

This will populate `mycol` with the value of `Aye`, a slash, the value of `Bee`, a colon and finally the value of `Sea`, i.e. / or : in a concatenation like this acts like an attribute whose value is a slash or colon.

These constructs can be used with any column type, however it usually only makes sense for strings.

You can also specify a `-` character to extract information from attributes which have values of the form `name=value`.

```
table fred
  column:INTEGER mycol=Aye-
endtable
```

This will remove any string from the start of the attribute up to =

For example, if the attribute `Aye` has the value `name=7`, then this will set the value of `mycol` to `7`. The `-` characer can also be used to remove several sets of `name=value` pairs. The first - removes up to '=' inclusive, the second up to ',' inclusive, the third the next '=' the fourth the next ',' and so on.

For example:

```
 table fred
   column:INTEGER mycol=Aye---
 endtable
```

Now if the value of `Aye` is `name=blah, name1=9`, it will be converted to `blah, name1=9` as the first minus is processed, and then the second one will be `9`. In both cases, if `Aye=9`, then the trailing minus will have no effect (because the search looks for a '=')

You can combine all of the above, for example:

```
table fred
  column:VARCHAR(25) mycol=Aye-+/+Bee
endtable
```

This will apply one '-' edit to the value of `Aye` to remove a prefix, append a '/' to it and then append the value of `Bee`.

If an attribute is not found, then the corresponding column is set to `NULL`, and a warning message generated. To suppress the message if you known that there is no corresponding LDIF attribute, then you may map the column onto the `NULL`value, using the special keyword `NULL`:

```
table fred
  column:REAL mycol=NULL
endtable
```

5.2.1 PRIMARY KEY COLUMNS

A column which is part of the primary key of a table cannot contain `NULL`, so its value must be specified in the LDIF. To specify that a column is part of the primary key for a table it can be declared with the following syntax:

```
primarykey:<type> <column>=[<attributeName>]
```

This is the same as `column`, except that if the attribute isn't present then GIN writes a warning message and doesn't update the table (or any multivalued link tables associated with it - see `multivalued`).

5.2.2 MULTIVALUED COLUMNS

In LDIF, some attributes may have multiple values. In order to map this on to the relational model used by R-GMA, it is necessary to create a new 'multivalued link table' to hold the multiple values. To configure Gin to publish multivalued columns, the following syntax can be used:

```
multivalued:<type> <column>[=[<attributeName>]]
```

Again, this is a direct replacement for `column` , but indicates that the column may be multivalued. The effect is to create a new table called `<column>`, containing two columns - one named `<table>UniqueID` and one named `Value`. The table must therefore contain a primary key column called `UniqueID` in order to have multivalued link tables associated with it.

Example:

```
table fred
  primarykey:VARCHAR(250) UniqueID
  column:INTEGER mycol
  multivalued:REAL mymulticol
endtable
```

This will create two tables: `fred`, and `mymulticol`. `fred` will have two columns: `UniqueID` and `mycol` , whereas `mymulticol` will have columns `fredUniqueID` and `Value` . The second table will contain all the values of `mymulticol`(which are of type REAL) found in the LDIF data.

5.3 INFORMATION PROVIDER OPTIONS

5.3.1 INFORMATION SOURCES

```
run <program to run to get LDIF information>
```

Specifies a command to run to interrogate the LDAP service provider. The command should return information in LDIF format on its standard output. More than one command can be specified using multiple `run` keywords (on seperate lines) - the information from all the commands will be combined.

```
staticinformation <table> \"<sql INSERT command>\"
```

`table` will be updated with specified command at the slow rate, i.e. as if it is a `slowtable`. **N.B. The table name appears twice in the line.**

Example:

```
staticinformation info \"INSERT INTO info (NAME, ADDRESS) VALUES ('ahost.com', '1.2.3.4')\"
```

5.3.2 PUBLICATION INTERVALS

`checkinterval <number of seconds>`

Mandatory keyword specifies the number of seconds which must elapse between updates. [1]

`maxpublishinterval <number of seconds>`

If this parameter is specified, tuples will not be published at the frequency specified by `checkinterval` unless they have actually changed. However, all tuples will be published at the specified maximum publish interval, whether they have changed or not.

For example if you set `checkinterval` to 30 seconds and `maxpublishinterval` to 300 seconds, the information will be checked every 30 seconds, and any tuples which have changed will be published. The full set of tuples will be published every 300 seconds, regardless of whether they have changed. This enables Gin to keep rapidly changing information up to date without flooding consumers with large numbers of identical tuples.

Note that tuples belonging to tables that do not have a primary key defined will be published on every update cycle since without a primary key it isn't possible to tell if a tuple is 'new information' or not.

If `maxpublishinterval` is not specified, all tuples are published on every update cycle, changed or not.

`slowtable <tablename>`

Specifies a table for 'slow' updating. Tuples belonging to a slow table are still updated at the specified check interval if their information has changed. However they may only receive a full update (changed or not) less frequently than non-slow tables. How much less frequently is specified by the `slowtablecounter` parameter.

`slowtablecounter <count>`

Number of iterations that slow tables are to be updated. For example, if set to 2, slow tables are only given a full update on every other cycle.

If not present, slowtables are treated as normal tables

5.3.3 DATA TRUNCATION

`truncate true|false`

This parameter specifies whether character data should be truncated to the maximum size allowable (`yes` or `true`), or if tuples should be rejected if the data in a character column is too long (`no` or `false`).

If the parameter is not specified, character data will not be truncated.

[1] This parameter used to be called delay. The delay keyword can still be used as an alias but it is now deprecated.