GUS User's Guide

Michael Saffitz

GUS User's Guide

Michael Saffitz

Table of Contents

List of Examples

Note

This document is in the early stages of development.

Chapter 1. Using GUS Plugins

Plugin Overview

Plugins are small programs that work with the GUS application to load data. See the Developer's Guide for more information about the Plugin API and writing plugins. GUS comes with two types of plugins: *Supported* and *Community*. Supported plugins have been tested in both Oracle and Postgres and have been confirmed to work properly, are portable across any site with a standard GUS installation, and widely useful, and meet the plugin standard outlined in the Developer's Guide. Community plugins are provided as part of GUS as well, but they fail to meet at least one of these criteria and they should be reviewed prior to use.

Registering Plugins

GUS requires that all plugins be registered prior to use. This allows for close auditing of the environment in which the plugin operated and ensures that changes within a plugin do not go unnoticed. When you attempt to use a plugin that has not been registered, `ga` will respond with an error:

```
USER ERROR: GUS::Supported::Plugin::InsertControl has never been registered.
Please use 'ga +create GUS::Supported::Plugin::InsertControl --commit'
```

You must then use the command provided to register the plugin. In this case:

```
$ ga +create GUS::Supported::Plugin::InsertControl --commit
```

If a plugin changes since being registered, you may receive a similar error, requesting that you use the `ga +update ...` command.

Running Plugins

Plugins are run using the `ga` command with the full name of the plugin, as well as any command line arguments:

```
$ ga GUS::Supported::Plugin::PluginName --argumentName  argumentValue
```

When you are ready to save the changes to the database, you must add the `--commit` flag to the `ga` command:

```
$ ga GUS::Supported::Plugin::PluginName --argumentName  argumentValue --commit
```

Creating new Plugins

For more information on how to create new plugins or modify existing plugins, please see the GUS Developer's Guide.

Chapter 2. Configuring GUS

Users, Groups, and Projects

GUS provides basic functionality for tracking changes and controlling access by users, groups, and projects. To use this functionality, you must first create User, Group, and Project entries in the database. If you've just completed installing GUS, these will be entered using the `dba` user and group, and the `Database administration` project (these three values are the default in every GUS installation). These tables are populated with either the LoadRow or LoadGusXml generic plugins. Examples are below.

Example 2.1. Adding a User with LoadRow

```
ga GUS::Supported::Plugin::LoadRow --tablename Core::UserInfo --attrlist \
        "login,password,first_name,last_name,e_mail" \
        --valuelist jsmith^^^p@ss^^^Jane^^^Smith^^^jsmith@example.org --commit
```

Example 2.2. Adding a Group with LoadRow

```
ga GUS::Supported::Plugin::LoadRow --tablename Core::GroupInfo \
        --attrlist name --valuelist samplegroup --commit
```

Example 2.3. Adding a Project with LoadGusXml

The XML to load is shown above, and is assumed to be named `sampleproject.xml`. The command to load the XML is shown below.

```
<Core::ProjectInfo>
     <name>Sample project</name>
     <description>A sample project</description>
</Core::ProjectInfo>
//
```

```
ga GUS::Supported::Plugin::LoadGusXml --filename sampleproject.xml --commit
```

Once these tables have been populated, you must either update your `$GUS_CONFIG_FILE` and `gus.config` file (in the `$PROJECT_HOME/install/` directory), or use the values from the command line when running plugins with the `--user`, `--group`, and `--project` flags.

External Databases

GUS supports tracking third party databases and third party data releases through the `SRes.ExternalDatabase`, and `SRes.ExternalDatabaseRelease` tables. These tables may be populated using the `InsertExternalDatabase` and `InsertExternalDatabaseRls` plugins.

Table and Schema Metadata

Metadata for tables is held in `Core.TableInfo`. Metadata for the Schemata are held in `Core.DatabaseInfo`. See the Developer's Guide for more information about extending GUS and adding to these tables.

Chapter 3. Building Websites with the WDK

The GUS Web Development Kit (WDK) provides a rich environment for rapidly creating query-centric websites. For more information on using the WDK, please see the online documentation [http://www.gusdb.org/wiki/index.php/GusWdk]

Appendix A. Troubleshooting and Getting Help

Troubleshooting GUS

Plugins have several options for increasing the verbosity of log messages. Add the `--help` flag to the end of your call to `ga` to see options.

Troubleshooting Resources

An active community of GUS users and developers is available on an informal basis to assist with using GUS. Posting to the GUS Mailing List [http://lists.sourceforge.net/lists/listinfo/gusdev-gusdev] or searching the archives [http://sourceforge.net/mailarchive/forum.php?forum_id=9903] is the fastest way to get assistance. When posting to the list, you should try to be as specific as possible, including the various command line arguments and full error messages, as appropriate. There are additional resources available on the GUS Website [http://www.gusdb.org] and GUS Wiki [http://www.gusdb.org/wiki].