Fairlight OneGate

Version 5

Documentation

Current Version: 05.01.00
Released: 08/31/12

Table of Contents

New Features in OneGate v5

Fairlight Consulting is proud to announce that Fairlight OneGate 05.00.00 has been released! Some of the new and exciting features in this version include:

- Support for templates, including fully recursive file, command, and field inclusion
- Support for email notifications upon errors
- Support for a more extended logfile format with more debugging and tracking information
- Support for dictating which program set to run via the path info segment of URLs, for use in integration with credit card gateways and other services where you have no control over fields returned, but you can control the URL

...And a host of other little improvements, minor bug fixes, and tweaks.

OneGate v5 is a drop-in replacement for OneGate 4.x and earlier. It is fully backwards compatible, requiring no reconfiguration.

Introduction

OneGate is an enterprise-class universal Common Gateway Interface (CGI) engine that allows communication between web browsers and "applications" of any kind.

The CGI specification defines a mechanism for encoding and transferring the data on the browser end, and decoding and parsing the data on the server end of a transaction. Originally, CGI was what

it is called, a "gateway" to another application. More and more programs have been written with their own parsing routines to accommodate the standard, as well as include the code to do the actual work desired. Many of these applications are designed exclusively for use on the web.

OneGate goes back to the gateway approach and generalises it. It is a configurable gateway that allows you to focus on writing and using your basic application, while making it easy to enable it for use on the web.

Instead of using many different task-specific gateway programs (one for emailing contents of a form, one for populating a database, one for running votes, etc.), OneGate lets you use one engine to drive *all* of these tasks and more, conveying the submitted data to your application via XML in a file stored to local disk on the server. If your application can handle XML data, you can utilise the submitted data as your needs dictate, and write a response file to be sent back to the client.

"Application" is a broad term in the sense that OneGate allows you to call as many commands as you need within your processing of the CGI request. You can call a database to store the data, then an email program to send the data to a recipient, followed by a program to generate a nice graph based on the data, if that's what you need. All these components, which I refer to as a *program set* are what I consider part of any given "application". The word application does not denote a single call within the context of OneGate.

Unlike other engines, OneGate is not limited to sending HTML pages back to the client. If you want, you can define the MIME type of your response file and send back a graphic, a sound file, a compressed archive, or whatever else is needed.

OneGate is also scalable in the sense that you may limit how many instances are allowed to run concurrently. If you interface to software that has only a five user license, you can allow only five instances of OneGate to run concurrently, and it will attempt to try for a configurable amount of time before giving up. You can author the HTML page that is returned if OneGate is forced to abandon a request due to user limits.

OneGate will also track use of your applications, logging when they were used, from what hosts, which application was requested, and what the end result of the request was.

OneGate also supports HTTP File Upload, and you can mix and match regular data and file uploads at-will, including multiple file uploads at once. If your application needs to receive whole files, OneGate fits the bill. The files are stored separately, and the information about the (safe) filename and the original remote filename are passed to your application within the XML data set that describes the query submission.

With simple yet powerful configuration, OneGate can ease the migration of existing applications into web-enabled functionality, or you can design powerful applications for use with OneGate from the ground up, without having to worry about the CGI getting in the way.

Additionally, OneGate allows you to access cookies as easily as you can the other CGI data.

On the security front, besides the usual precautions that all CGI programs should take, OneGate provides for a virtual application firewall for each specific application, capable of denying and allowing both wildcarded subnets and specific IP addresses. Also available for each application is the ability to limit use by user and/or group. Both the subnet and IP restrictions and the user authentication are configurable at the application level, meaning you can have different sets of rules for each application. For an inventory listing, you could let anyone access the application except abusive sites. For data entry into the inventory, you could limit access to only machines on the VPN of a multi-interface machine, and you could even allow only certain users from those machines. Unlike

traditional firewalls, you allow or deny access to each application, not the whole machine or site--all without having to enlist the help of your network or system administrators. The security is as flexible as you want it to be.

For those situations where you must temporarily disable either individual applications, or all applications running under OneGate, you may disable it at either level with a single message file with which you also notify the users of the situation. This functionality is great for evening backup periods and the like.

OneGate has been further expanded to conform with AJAX methodology for more robust web development.

Features

- Supports both straight CGI and AJAX
- Supports multiple program sets (applications)
- Supports GET, POST (x-www-application-urlencoded, or multipart/form-data)
- Supports HTTP File Upload with safe filename handling
- Provides output from the web to your application in XML with a well-defined DTD
- Supports optionally saving TEXTAREA field contents to separate files rather than contents in the XML output, accommodating programs that cannot handle extremely long internal fields, or schemes where saving such data separately is desirable
- Supports unlimited commands within each program set
- Supports access logging (unlimited, rolling, or disabled)
- Supports session-count limiting
- Supports user-defined environment variables, at both master and application levels
- Provides access to cookie data
- Provides custom error page facilities that let you match the style of your site
- Allows you to send back any type of data file, using any MIME type
- Provides access to the input, output, and MIME-type filenames for all commands via magic cookies
- Designed with system security in mind
- Provides application-level virtual firewalling, allowing you to deny or allow wildcarded subnets or explicit IP addresses
- Provides application-level user authentication, which cooperates with HTTP Basic Authentication, or can be used strictly internally
- Supports multiple testbeds as well as production configurations
- Supports a template system, utilising file, command, and (sanitised and safe) field inclusion in the output stream
- Supports mailing one or more addresses with error notifications
- Supports easier integration with credit card or other payment gateways

- Includes a text-based, menu-driven configuration tool
- Includes a cloning tool for expedient "cookie-cutter" configuration copy-and-alter development
- Source code immediately available for modifications, if necessary or desired

Requirements

This program requires Perl version 5.8 or higher to run. It also requires the Perl CGI module to be installed on your system to operate correctly. (Technically, OneGate v5.x and higher can still run on versions of Perl below v5.8, but you must have the Net::SMTP module installed. This version became part of the core distribution in Perl v5.8, which is why it's listed as the default requirement.)

You will also need a web server configured to allow CGI execution.

OneGate has been tested and run on Solaris, Linux, SCO UNIX OpenServer 5.0.6 and 6.x, AIX, Windows 2000, and Windows 2003/2008 Server. OneGate has been tested and run under IIS 6 as well as various versions of Apache. It should encounter no difficulties on any platform on which perl will compile and run correctly.

Licensing

It should be noted that there is no demo version of OneGate, nor will there be a "Lite" version.

We will provide whatever pre-sales assistance is required to demonstrate whether OneGate is the right product for your needs. If you have questions or comments, please direct them to **sales@fairlite.com** without hesitation, and we will answer as promptly as possible.

We realise that software is an investment, and hope that you preview the documentation available for this product and make sure that it will suit your needs before purchasing a license. We also hope you avail yourself of the opportunity to ask any pre-sales questions you might have. All license sales are final and non-refundable.

OneGate is licensed at a list price of **$995.00 USD** *per server on which it is installed*. Bulk discounts may be negotiated for purchases of more than five copies at the same time.

Each license fee entitles you to use OneGate on one server, in any role you require. You may modify the program to further suit your needs, and are under no obligation to release changes back to Fairlight Consulting. However, derivative works and/or modified versions **may not** be resold or otherwise distributed. Similarly, you may not copy OneGate, modify it, and run the altered version on another machine. You must purchase another license to use it in any form, altered or otherwise, on an additional machine. You may use an altered version and the original version on the same machine under a single license, however. ***"Machine" shall be defined as one instance of an operating system, for the purpose of this license. Machines which run multiple concurrent operating systems (virtual machines) count as multiple machines, and require additional licenses for each instance.*** The licensee agrees to keep the source code confidential and protected.

Upon receipt of payment for a license, access to the program will be generated for the licensee, and such information shall be delivered to the email address associated with the PayPal payment.

Upgrades for the product were previously free from initial release through version 03.00.03. Past that point, upgrades became paid upgrades between major versions. Upgrades from 4.x to 5.x are a paid

upgrade. Please contact Fairlight Consulting regarding any upgrade requests. Any upgrades between minor versions in the 5.x tree will be provided free of charge. Fairlight Consulting reserves the right to charge upgrade fees at major version changes.

There is ***no warranty*** for this software. This software is offered **"AS-IS"** and without warranties as to performance or merchantability or any other warranties, whether expressed or implied.

Good computing practice dictates that any program should be thoroughly tested with non-critical data before deploying it for production. The user assumes the entire risk of using the program. In no event shall Fairlight Consulting be held liable for loss of data, failure of performance, or any other damages, be they real or perceived.

Support

Extended (non-bug-related) support for OneGate is available at our standard hourly rates. Because we offer pre-sales assistance in determining if OneGate is right for your needs, and because documentation is readily available, anything not covered by either of these is deemed an at-cost support issue.

You may request *any* kind of technical assistance with OneGate by sending email to **onegate@fairlite.com**, including bug reports and feature requests.

Feature requests may be commissioned for special functionality, if desired. Any non-commissioned requests are subject to being implemented solely at the discretion of Fairlight Consulting. This may include not being implemented at all, depending on how useful we think the feature would be in general. Commissioned requests can be price-negotiated based on whether the features requested are allowed to be re-integrated into the main product, or whether they shall remain proprietary and exclusive to the commissioning party.

We also offer consulting on how to achieve specific results using the product through this support mechanism, and would be happy to assist you in this regard.

Program Flow

The logic flow of CGI using OneGate is straightforward, and is as follows:

1. A request is sent from a browser or other application to the web server (httpd) at the URI that indicates the location of the *onegate* program. This request contains one hidden field called *onegate_set*, which tells OneGate which program set to concern itself with for this particular execution of the program.

2. The web server (httpd) then starts *onegate*.

3. OneGate performs various security checks. Assuming that everything about the data and configuration is successful, and that the execution is not aborted due to a configurable session count or security-related configuration, it writes the XML file into a spool directory, and then executes all commands listed in the program set, in order. If the session count was exceeded for the configurable amount of retry time, a user-authorable error page is sent back instead at this point, and the program terminates.

4. After all programs in the set have been executed successfully, the MIME type file is checked for in the spool. If one was written by your application, and is valid, it determines the MIME type of the outgoing response to the server (httpd) for delivery to the remote client.

   The result file, which at least one (if not several) of your commands in the program set should have written to in the spool directory, is then passed back to the remote client via the web server (httpd) with any template inclusion and substitution handled inline. In the event of a command failure, a OneGate warning is issued to the client

instead. Due to security considerations, this is the case if *any* command in the set fails (as determined by an exit code other than zero). This is the default error handling, although v4 of OneGate added conditional command execution, which is detailed later in this documentation. Mail may be configured to be sent upon errors.

5. OneGate removes all spool files, including any uploaded files which your application should have processed, logs the access, and terminates.

DTD

The OneGate DTD for incoming parsed form data is as follows:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE form_submission [
<!ELEMENT form_submission (form_field*)>
<!ELEMENT form_field (field_name,field_value+)>
<!ELEMENT field_name (#PCDATA)>
<!ELEMENT field_value (#PCDATA)>
<!ATTLIST form_field type (file|value|cookie|textarea) "value">
<!ATTLIST form_field orig_filename CDATA #IMPLIED>
<!ATTLIST form_field size CDATA #IMPLIED>
]>
```

There are several things to note regarding how OneGate presents data:

- All fields that are submitted will be reported, even if empty. In the case of empty fields, the *field_value* element will be an empty container.
- All regular fields are assigned **value** for the *type* attribute to the *form_field* element. All file upload fields are assigned the type **file**.
- File uploads result in the passing of the remote filename in the *orig_filename* attribute of the *form_field* element. The file size in bytes is passed in the *size* attribute. The safe, local filename is contained in the *field_value* element. The *type* attribute to the *form_field* element will be assigned the value of **file**.
- In the event that a file upload field is present but not utilised, the *orig_filename* and *size* attributes will be empty strings, and the *field_value* element will be an empty container.
- Textarea fields (ones set explicitly with the **TEXTAREA** option) will result in the *field_value* element containing a path to a safe filename rather than the contents of the entire TEXTAREA. This file will contain the actual contents of the submitted TEXTAREA. The *type* attribute to the *form_field* element will be assigned the value of **textarea**. Textarea fields which are empty will result in an empty *field_value* container.
- Cookies will be presented as form fields, and assigned the *type* **cookie**, which will differentiate them from regular form values. Cookies which are requested but not available will result in the *field_value* element being an empty container.
- All *field_value* contents will be encapsulated inside the **<![CDATA[]]>** syntax, so as to preserve any possible SGML input that might result from form data. In the case of empty containers, the CDATA block will not be present, and the container will truly be empty.

Installation

Installing OneGate is very straightforward, and involves a few simple steps.

OneGate's main program, *onegate*, should reside in a directory that has CGI execution enabled in your httpd configuration. The program's working directory, also called *onegate* should reside somewhere **outside** of the web document space on your server for reasons of security.

If you are installing on *nix, make sure you log in as or su to the same user as should be running onegate, prior to installation or any (re)configuration. If you do not, you will need to manually adjust permissions and ownerships accordingly.

To install OneGate, follow these steps:

1. Copy the distribution tarball to a directory where you have write permission.
2. Untar the distribution tarball.
3. Change directory into *onegate-install*.
4. Run **perl ./install.pl** to begin the installation process.
5. Answer the location questions.
6. Adjust the ownership of the files and directories under OneGate's directory tree if necessary. (See *Dealing with UID Conflicts* in this documentation for a more thorough explanation.)
7. Proceed to the section on configuration.

Upgrading

To upgrade OneGate, follow the steps for installation.

Please note that OneGate's installer will overwrite the old program file, but your entire *onegate* directory tree will remain untouched by the process. All existing configuration will be retained.

Despite this fact, good computing practice dictates that one should make a backup of one's configuration before performing an upgrade. It is not necessary, but it *is* wise.

Configuration Instructions

In this section we will walk you through the relatively simple tasks involved in configuring OneGate for use with your applications. There are twelve sections in all.

Before we begin, it should be stated that in the following sections, pathnames will contain certain parts in square brackets. These parts are:

- [maindir] - The full path to the main *onegate* directory for configuration and spooling.
- [progset] - The name of a valid program set, used as a directory name.

It is a good idea to familiarise yourself with manual configuration. After doing so, you can either configure OneGate manually, or you can use the ***ogconfig*** configuration tool. The choice is yours, but you *should* understand the configuration at its basic level before using the tool.

Valid Program Set Definitions

Program sets are single word names that tell OneGate which application it is working with at any given time. A program set name may contain only alphanumeric characters, and the underscore symbol.

When called, OneGate will verify the program set name passed to it in the form field *onegate_set*. The file used to verify validity of program set names is *[maindir]/valid.sets*. This file should contain one program set name per line. Blank lines or lines starting with a pound sign (comments) are ignored. You can *not* put comments on the same line as a program set name. This file **must** be present.

Main Environment

Global environment variables that all your applications require may be set by using the *[maindir]/main.environment* file. The presence of this file is optional.

The format of this file is identical to Bourne shell syntax, with one **VAR=value** assignment per line. There are a few exceptions to the Bourne-like quality of the syntax. Exports are not needed. OneGate handles this for you, as the file is not processed by a shell. Comments may be on individual lines, and blank lines are ignored. You may *not* put comments after value contents. The other notable difference is that you do not need to quote your values. Any quotes included (be they single or double) will be literally interpreted as part of the value for your variable.

Main Configuration

The file *[maindir]/main.config* controls global behaviour for OneGate's handling of all applications. The presence of this file is optional, and if it is absent or empty, default values will be used.

There are 18 options in all, which are described in detail on the *Configuration Options* page. The format of this file is one **option=value** assignment per line. Comments may be on lines by themselves, and blank lines are ignored.

Global General Error File

The file *[maindir]/error.page* may be used to create a "pretty" general error page in the event OneGate encounters an error during execution. This page should be in HTML format, and is intended in the same vein as *httpd* custom error pages.

Using this file, you can suppress OneGate's normal, terse error reporting and give your users an error page that is integrated closely with the look and feel of your site, and allows for graceful redirection at your discretion.

If the *devel_debug* option is enabled, custom error pages are disabled.

Global Field Definitions File

The file *[maindir]/main.fields* controls which fields OneGate will accept globally for all program sets, and their definitions. See the Program Set *Field Definitions* page for full details on syntax.

Any fields defined in program sets automatically override the fields defined in this file in the event of a redefinition conflict.

Global Cookie Definitions File

The file *[maindir]/main.cookies* controls which cookies OneGate will accept globally for all program sets, and their definitions. See the *Program Set Cookie Definitions* page for full details on syntax.

Program Set Field Definitions

The file *[maindir]/configs/[progset]/fields* is used to describe the permitted and required fields for each program set. This file **must** be present, although it may be empty if your program requires no fields.

The format of this file is one entry per line, consisting of three fields separated by whitespace. The fields, in order, are:

1. **fieldname** - The literal field name used in your form.
2. **mandatory** or **optional** - Whether the field *must* be present at submission. Please note that this means exactly what it says, that the *field* must be present, **not** that the field must contain data. This setting is not meant to catch empty fields; it is meant to prevent abuse by people faking submissions and omitting fields, while at the same time allowing you to make certain fields (like file uploads) optional for debugging or other purposes. Any field set to mandatory must be present at submission, or OneGate will return a *Missing Fields* error and terminate the transaction. Note that if you utilise the *allow_field_wildcards* option, optional fields may be wildcarded to cut down on repetitive definitions.
3. **FILE** or **TEXTAREA** - If this optional field is present and set to **FILE**, it denotes that the field is an HTTP File Upload field, and should be treated as such. In this case, the uploaded file is saved to a safe filename, and the contents of the XML output are adjusted as detailed in the description of the *OneGate DTD* All file upload files are deleted after your program set finishes running, so you must copy them to somewhere for permanent storage if you wish to keep them. If this field is present and set to **TEXTAREA**, it denotes that you wish to store the field contents in a separate file and only reference the path to the file in the XML output, as specified in the *OneGate DTD*. All TEXTAREA files are deleted after your program set finishes running, so you must copy them to somewhere for permanent storage if you wish to keep them. Absence of this field is the same as giving it any value other than **FILE** or **TEXTAREA**, and will result in the field being treated as a regular value field.

Please note that the main OneGate field *onegate_set* is always accepted, and does not need to be in the field definition file for any program set.

Additionally, the special fields for user authentication (if you enable it) are always accepted, and do not need to be in the field definition file for any program set. These fields are *onegate_user* and *onegate_pass*.

Any other field not listed in this file will cause the entire request to be denied with an error.

NOTE: A leading cause of "Extra fields given" errors (error 12) is forgetting to include the field for your SUBMIT or IMAGE buttons. This field should ideally be defined as *optional*, as it will not always be returned. If the browser allows a user to submit a form by hitting <ENTER> in a text field or elsewhere, the submit button value may not be returned. If the button is pressed, it should be returned. Hence, *optional* is the advised setting for these types of fields.

Program Set Cookie Definitions

The file *[maindir]/configs/[progset]/cookies* is used to describe the requested cookie names to be retrieved for each program set. The presence of this file is optional. The syntax allows for blank and separate comment lines. Each cookie name to be retrieved should be listed on a separate line.

Program Set Command File

The heart of your interaction with OneGate, the file *[maindir]/configs/[progset]/program.set* controls what commands are executed within your application. This file **must** be present, and configured with the list of your commands for the program set.

The syntax allows for blank and separate comment lines. You may *not* place comments on the same line as a command. Each command should be on a separate line, and each will be processed in order at execution.

You may use any valid command on your system, and shell metacharacters (redirects, etc.) *are* allowed. Please note that OneGate clears the *$PATH* environment variable at the beginning of its execution, so you effectively have no path unless you set one in either the main or program set environment variable files. If you do not define a path, you will need to explicitly state full paths to your commands.

There are seven "magic cookies" that allow you to give your commands access to the relevant files and data associated with a OneGate transaction. Wherever these tokens appear in your commands, they will be replaced with the appropriate pathname.

- **%i** - This token represents the XML data file generated by OneGate. This file contains your form data.
- **%o** - This token represents the output file that will be sent back to the remote client after all commands in the program set have successfully been executed.
- **%t** - This token represents the Reply MIME Type File, which allows you to control the MIME type with which your response is sent.
- **%f** – This token represents the file which handles individual field injections.
- **%n** - This token represents the internal lock number OneGate uses for your current transaction.
- **%p** - This token represents the program name to call in your URL's when using the *multiple testbed functionality*.
- **%u** - This token represents a unique ID that can be used by both your applications and commands in your program set to let you save and utilise temporary files between commands. (e.g., *%u-sync.sql*) **NOTE: Any files you create are created in *[maindir]/spool/[progset]/tmp* unless you specify a full path. Should you create any "external" files, you should remove them manually from within your program set if you wish them to be deleted as part of the transaction, as OneGate will *not* remove these non-OneGate-specific files automatically.**

Should you be unable to utilise the command line to pass filenames or the internal lock number to your application, an alternate method is available in the form of environment variables. These do not go in the configuration file, but instead may be accessed from within your application:

- **ONEGATE_INFILE** - Identical to the file that **%i** denotes in the program set command file.
- **ONEGATE_OUTFILE** - Identical to the file that **%o** denotes in the program set command file.
- **ONEGATE_MIMEFILE** - Identical to the file that **%t** denotes in the program set command file.
- **ONEGATE_FIELDFILE** – Identical to the file that **%f** denotes in the program set command file.
- **ONEGATE_LOCK_NUMBER** - Identical to the number that **%n** denotes in the program set command file.
- **ONEGATE_CGINAME** - Identical to the program name that **%p** refers to in the program set command file.
- **ONEGATE_UNIQUE_ID** - Identical to the program name that **%u** refers to in the program set command file.

*NOTE: As of v04.00.00, the **%t** and **ONEGATE_MIMEFILE** methodology is not required. The new mime_type option in the configuration files will suffice. If the legacy functionality is used, it will override whatever may be set in the configuration file(s).*

Commands may be run conditionally as of version 04.01.00. The syntax for doing so involves prefixing your commands with certain characters at the very beginning of the line. The following syntaxes have the following effects on your commands:

- **command** - Without modifying characters at the start of the line, your commands will run in regular mode. This means that if they exit with a non-zero status (indicating a failure in most cases), OneGate will exit with Error 17 after dispersing the appropriate error page.
- **!command** - With an exclamation point prefix, if the current command exits with a non-zero status, subsequent commands continue to be executed.
- **?command** - With a question mark prefix, the current command is **only** executed if the *immediately preceding command* exited with a non-zero status. This can be used to institute "fallback" commands in case other commands fail.
- **?!command** - With a question mark and *then* an exclamation point, the command will **only** be executed if the *immediately preceding command* exited with a non-zero status, *and* subsequent commands will continue to be executed if the current command exits with a non-zero status. **NOTE: The "?!" sequence is *not* reversible. A syntax of "!?" will *not* do the same thing, and should not be used.**

Additionally, any command may create a file equivalent to **%u.STOP** in the current working directory (that is, a file *[maindir]/spool/[progset]/tmp/[ONEGATE_UNIQUE_ID].STOP* where the unique ID is the value from that environment variable or taken from **%u**) in order to ***entirely halt*** execution of further commands. Before each command is executed, a check is made for the presence of this semaphore file. If it is present, no further commands will be executed. Thus, it is possible to have a file "fail" programmatically, write an output page, and use this semaphore to cause no further commands to run, in order to get that error page to the browser and proceed no further.

Program Set Environment

Similar to the main environment variable file, you may set environment variables for each specific application independently by using the *[maindir]/configs/[progset]/environment* file. This file is optional, and if it is empty or absent, global and then default values will be used, in that order of precedence.

The format of this file is identical to that of the main environment variable file. Please note that any identical variable names in the application environment file will *override* global entries of the same name.

Program Set Configuration

Similar to the main configuration file, you may set the OneGate options at the application level with the file *[maindir]/configs/[progset]/config*. This file is optional, and if it is empty or absent, global and then default values will be used, in that order of precedence.

As with environment variables, any entry at the application level will override the global settings. The sole exception is the *max_session_count* option, which will be ignored in the application context and is only valid at the global level.

Program Set General Error File

The file *[maindir]/configs/[progset]/error.page* may be used to create a "pretty" general error page in the event OneGate encounters an error during execution. This page should be in HTML format, and is intended in the same vein as *httpd* custom error pages.

Using this file, you can suppress OneGate's normal, terse error reporting and give your users an error page that is integrated closely with the look and feel of your site, and allows for graceful redirection at your discretion.

Any program set general error file automatically overrides the global general error file, if the global file exists.

If the *devel_debug* option is enabled, custom error pages are disabled.

Lock Count Error Message File

The file *[maindir]/lock.error* should contain a valid HTML page to be returned in the event that OneGate encounters its maximum number of instances as set by the *max_session_count* option, and exceeds all retry time. Place any message you want in this file, in HTML format. Bear in mind that this is a *global* file, and will be returned on all occurrences of this event. This file **must** be present.

Program Set Virtual Firewall

The virtual firewall for an application is configured through the file *[maindir]/configs/[progset]/host.access* and is disabled if this file does not exist. Comments and blank lines are allowed in this file. The format of the file is one rule per line, consisting of the following fields in order, separated by whitespace:

1. An IP address. This can be a complete address, or a wildcarded address. Any of the four octets may be an asterisk as a wildcard for that octet. All four octets must always be present, even if you wild card all but the first two. For example: *192.168.*.** is correct, but *192.168* is incomplete and will cause OneGate to issue an error message regarding the misconfiguration of the address.

2. A class. Valid classes are **allow**, **deny**, or **alwaysdeny**.

3. A path to the file containing the HTML message to relate to the browser in the case where a ban is enforced. This field is only used (and necessary) if the class field is either *deny* or *alwaysdeny*. If the class is *deny* or *alwaysdeny*, this field **must** be present and point to a file that exists.

The order in which the rules are listed is not important as of version 04.00.00.

- All IP addresses from remote clients that match no bans are allowed by default.

- An IP address that matches a *deny* rule will be denied unless there is a subsequent *allow* rule that negates it. Given a remote address of *192.168.1.13*, a rule that denies **.*.*.** will be overridden by a rule that allows *192.168.1.**.

- An IP address that matches an *alwaysdeny* rule will **always** be denied, regardless of any *allow* rules it may always match. Given the above example, you could allow all of that subnet *except* the remote address by adding a rule for the specific IP address, of the class *alwaysdeny*.

Rules that are of the classes *deny* or *alwaysdeny* **must** have the third field which points to an HTML file that will be presented to the browser, explaining the ban. The file you choose to present is entirely at your discretion, but it will be presented with the *text/html* MIME type. The file listed **must** exist at the location indicated, or OneGate will generate an error noting that there is a missing subnet denial message file.

The virtual firewall stage is handled *before* the user authentication stage, if both are enabled.

Program Set User Authentication

The user authentication for an application is configured through the file *[maindir]/configs/[progset]/allowed.users* and is disabled if this file does not exist. Comments and blank lines are allowed in this file. The file denotes the users that *are* allowed to use the program set. The format of this file consists of two fields per line:

1. **user** or **group** - A description of the type of entry for the rule.
2. The user name or group name which is allowed by the rule.

The use of a group entry automatically allows all users that fall within that group.

If you create and utilise the *allowed.users* file, you must also create the files *ht.users* and *ht.groups*. These may be symbolic links to already existing files elsewhere on your system on *nix platforms. The formats of these files follow the standard formats that the Apache web server recognises:

ht.users format:
username:encryption_string
ht.groups format:
groupname: user user user …

The *ht.users* file, should you not have one already created, may be created and maintained with the program *htpasswd*, which should be installed on any server with a web server present.

The *ht.groups* file, should you not have one already created, is created by hand. Please note that as with Apache, additional group lines with the same group name will append their users to the overall list of legal users for that group name, rather than replace the previous list. Thus, you can break up the list of users for large groups over multiple lines starting with the same group name, for increased legibility.

The user authentication may be utilised in two ways:

- The *REMOTE_USER* environment variable is checked first. If a user has already validated themselves with httpd within the same subdirectory tree that the OneGate program resides *and that authentication was for the same request type, either **GET** or **POST**, whichever you are using with your application*, then if the user is in the allowed.users table, and exists in the *ht.users* file, the user is allowed to use the program set. Please note that if a user has previously authenticated for **GET** but *not* for **POST**, and you are using **POST** in your submission (or vice-versa), the *REMOTE_USER* environment variable will ***not*** be set! You must previously authenticate for the same type of submission as you are using for your program set in order to utilise the pre-authentication.
- If the *REMOTE_USER* environment variable is not set, two special fields are checked for. These fields are *onegate_user* and *onegate_pass*. If you supply these (either from INPUT TEXT and INPUT PASSWORD fields that a user can fill out, or in HIDDEN fields generated from another program set [or even another source entirely]), OneGate will perform the encryption check on the password internally and attempt to validate the user against the configuration you set. **For security reasons, *do not* put the *onegate_pass* field in a static file on your web server! Anyone can view the source of a form, and your password would be present in plain text, thus compromising your security.** This method is meant to be used in a single session, with the fields either filled out manually or generated by software in dynamic forms that do not reside on disk.

The *allowed.users* file is OneGate's equivalent to Apache's *.htaccess* file, if you care to think of it that way. The difference (besides being less complex) is that any users not accounted for in this file will not be permitted to use the program set. There is no need (or ability) to manually deny anyone access. All listed users (or users included in listed groups) will be authenticated. Anyone else is denied access.

In addition, if you activate user authentication, you **must** create the file *[maindir]/configs/[progset]/invalid.user.message* for display to the remote browser. This file will be presented with a MIME type of *text/html* when a user is denied access.

The user authentication stage is handled *after* the virtual firewall stage, if both are enabled.

"No Login" Disabling Mechanism

OneGate may be told not to execute any applications (global) or a specific application (program set) by use of a *nologin* file. This file not only tells OneGate not to execute the usual functions, but the contents of the file are presented to the remote browser as an explanation of the refusal of service. The file will be presented with the MIME type *text/html*. The locations of the global and program set files are *[maindir]/nologin* and *[maindir]/configs/[progset]/nologin*, respectively. The global message will override any program set messages, should both be present simultaneously.

Configuration Options

The following options are valid for OneGate's configuration files:

wait_interval

When OneGate is activated, it performs a check to make sure that no more than a configurable number of instances of itself are running at the time. If it detects that number are already running, it will retry the condition a configurable amount of times. This option sets the granularity for these retries. The argument is an integer expression denoting seconds. The default for this option is **1** second. This option may be defined at the global and application levels.

wait_attempts

The companion to *wait_interval*, this option denotes the number of retries to attempt before giving up and sending the lock count error message file to the remote client. The formula for total time spent retrying is *wait_interval * wait_attempts*. The default for this option is **30** attempts. This option may be defined at the global and application levels.

uploads_allowed

This option dictates whether HTTP File Uploads are accepted. The option takes the words **yes** or **no**. The default is **no**. This option may be defined at the global and application levels.

max_upload_size

This option sets the maximum size in bytes to accept in a single uploaded file. This option takes an integer value or the word **unlimited**. The default setting is **102400**, which is 100K. This option may be defined at the global and application levels. Be careful about using the **unlimited** option needlessly, as someone could fill up your disk quite quickly. It is included for such purposes as doing web-based remote disk space hosting. This option may be set at both the global and application levels.

max_session_count

This option sets the maximum number of concurrent instances of OneGate that may run at once. It is intended to provide a "throttle" mechanism which may be used to guarantee that you do not exceed

an application's user license count, to limit the amount of "seats" of a license count that are eaten up by web application use, or to reduce load on the system for intensive applications. This option takes an integer argument. The default for this option is **128**. This option may ***only*** be set at the global level.

devel_debug

This option specifies whether OneGate should reveal the exact value of which fields are missing, extra, or misconfigured, and/or which commands fail on execution. This functionality is to be considered ***insecure*** for normal operation, and should *only* be used during development. When not actively being used for debugging, this functionality should be disabled to enhance security. When enabled, this option *disables* any custom error pages you have defined (global or application). This option takes the words **yes** or **no**. The default is **no**. This option may be defined at the global and application levels.

max_log_size

This option controls how OneGate logs access to program sets. If set to **unlimited** or omitted altogether, OneGate will log all access information. If set to **0** (zero), logging will be disabled. If set to any positive integer, logging will be constrained to that number of lines *per logfile*, and will roll the oldest entries off to make room for newer entries. The default is **unlimited**. This option may be defined at the global and application levels.

new_log_format

This option controls whether the original logfile format is used, or a new and more robust one is used. The new format includes both the session ID and any extended debugging information that is included when **devel_debug** is enabled. This option takes the words **yes** or **no**. The default is **no**. This option may be declared at the global and application levels.

mime_type

This option sets the MIME type of a successful response from the program set. This option may be set at the global and application levels. *In addition, please note that if the **%t** or **ONEGATE_MIMEFILE** are written to, this option is overridden by the value placed in that file, in order to preserve legacy functionality from configurations deployed before this option existed.*

lock_error_mime_type

This option sets the MIME type of the error message delivered when a session could not obtain a session due to system use exceeding the combined effects of *max_session_count*, *wait_attempts*, and *wait_interval*. The default is **text/html** in regular testbeds, and **application/xml** in *_ajax* testbeds. This option may ***only*** be set at the global level.

error_mime_type

This option sets the MIME type of an error response from the program set. This option may be defined at both the global and application levels.

ban_mime_type

This option sets the MIME type of the response given when a host is denied by the *host.access* file. This option may be defined at both the global and application levels.

access_mime_type

This option sets the MIME type of the response given when a connection is denied by the *allowed.users* file. This option may be defined at both the global and application levels.

nologin_mime_type

This option sets the MIME type of the response given when a connection is denied by the *nologin* file. This option may be defined at both the global and application levels.

sub_error

This option determines whether the *special token values* will be substituted into the response in event of an error response. The option takes the words **yes** or **no**. The default is **no**. This option may be defined at the global and application levels.

sub_success

This option determines whether the *special token values* will be substituted into the response in event of a successful program set run. The option takes the words **yes** or **no**. The default is **no**. This option may be defined at the global and application levels.

sub_includes

This option determines whether templating substitition is enabled or not. Please see the section on Templating Substitution later in the document for more details. The option takes the words **yes** or **no**. The default is **no**. This option may be defined at the global and application levels.

sub_fields

This option determines whether individual field substitution is enabled of not. Please see the section on Individual Field Substitution later in the document for more details. The option takes the words **yes** or **no**. The default is **no**. This option may be defined at the global and application levels.

allow_field_wildcards

This option determines whether fields may be wildcarded in the *fields* file for a program set. This facility allows you to specify one pattern a single time to allow many similar fields in one fell swoop. If turned on, this facility allows arbitrary portions of field names to be included **only in *optional* fields**. Wildcarding is done in the form of an asterisk. At the position of an asterisk, any valid field name characters may appear, but to not have to appear. Thus, the definition of an optional field "myfield*" would match "myfield", "myfield_23", etc. The option takes the words **yes** or **no**. The default is **no**. This option may be defined at the global and application levels.

cache

This option determines whether the header *Cache-Control: no-cache* is utilised. If set to **yes**, the header is not given to clients. If set to **no**, the header is delivered. The option takes the words **yes** or

no. The default is **yes** to maintain backwards compatibility with 3.x. This should be set to **no** for AJAX applications. This option may be defined at the global and application levels.

no_unlink

This option determines whether the temporary working files usually removed from the spool directory for the program set are removed at the end of the run. The option takes the words **yes** or **no**. The default is **no**. This option may be defined at the global and application levels.

basename_uploads

This option determines whether the original filename for file uploads is stripped to the basename or not. A setting of **yes** will strip the filename to its basename.. The option takes the words **yes** or **no**. The default is **no**. This option may be defined at the global and application levels.

mail_errors

This option determines whether emails will be sent with error information when errors occur. The option takes the words **yes** or **no**. The default is **no**. This option may be defined at the global and application levels.

mail_server

This option dictates the SMTP host which will be used to send mail. SASL authentication is *not* supported at this time. The option takes a hostname or IP address as its value, and must be defined if error mails are turned on. This option may be defined at the global and application levels.

mail_sender

This option dictates the sender address for error emails if **mail_errors** is enabled. The option takes an email address as its value, and must be defined if error mails are turned on. This option may be defined at the global and application levels.

mail_recipients

This option dictates the recipients for error mails if **mail_errors** is enabled. The option takes one or more email addresses. Multiple addresses are separated by a comma with optional whitespace. This option may be defined at the global and application levels.

Output Substitution Tokens

OneGate allows you to place tokens within your output, and within all error files. These tokens are replaced with the relevant data corresponding to each token. In this fashion, you can create static XML or other error files, or even substitute values from OneGate into your *successful* program output. This allows for enhanced development, debugging, and information conveyance.

The following substitution tokens are valid for OneGate's output, if the *sub_error* or *sub_success* configuration options are enabled with a setting of **yes**. If the relevant option is not enabled, no substitutions will be performed on the output.

#OGErrorCode#

This token will be replaced with the numeric error number encountered.

#OGErrorText#

This token will be replaced with the text error message corresponding to the error encountered.

#OGDevelError#

This token will be replaced with the extended information provided by **devel_debug** if that option is set to **yes**. If that option is disabled, the token will simply be deleted.

#OGVersion#

This token will be replaced with the OneGate version number of the running copy.

#OGProgram#

This token will be replaced with the formal program name and version (i.e., "Fairlight OneGate v04.00.00").

#OGSessionID#

This token will be replaced with the session ID if **devel_debug** is set to **yes**. If that option is disabled, the token will simply be deleted.

#OGGeneratedLink#

This token will be replaced with a hyperlink to the OneGate site and text "Powered by OneGate".

Templating Substitution

As of version 05.00.00, OneGate has a new “template” system of output available to it. If you're at all familiar with PHP include directives, or better yet, Server Side Includes in Apache, you'll recognise the power and versatility that this system offers. Using the template feature allows you to include static files, temporary files generated from within the program set, the output of commands not included in the program set, and even directly include field information that was submitted to OneGate, all within your output stream, without gathering it all up and amalgamating it first. This system is especially potent when used within environments (such as filePro), where having *all* your HTML, JavaScript, etc., inside another program is often quite painful, especially when performing modifications.

The first thing to understand about templates in OneGate is that you still write your normal output file. The difference is that the output file contents are probably going to be drastically smaller. There are two types of text in your output stream now. You have normal text, which is output directly to the client, same as it always has been. You now also have include directives, however. These directives serve as instructions for OneGate on what files, command output, or fields to substitute into the output stream in place of the directive before sending the output to a browser or other web client. Everything except include directives is passed unmolested to the remote client, so unless you turn on *sub_includes* and use one or more include directives, OneGate v5 is fully backwards compatible with OneGate v4.

The include directives are as follows:

FILE: /path/to/file [delete]

This directive will include the contents of the indicated file in place of the directive. If the file does not exist or cannot be opened, an error message will instead be substituted. If the optional **delete** keyword (sans brackets) is present, the file will be removed after being included. The deletion methodology is handy if you generate dynamic files, rather than simply using static templates. If you generate multiple files from your programming with filenames which use the **ONEGATE_UNIQUE** variable as part of their filename, you can do things like ##### *File: %u.tmp1 delete* #### and your temporary file will be removed after inclusion.

COMMAND: /path/to/command [arguments]

This directive will include the output of the indicated command in place of the directive. Exit codes are not trapped, but any errors written to standard output will instead be substituted.

FIELD: fieldname [optional separator]

This directive will include the contents of the CGI field as supplied to OneGate during a request. If the field was not supplied, a zero-length empty string will instead be substituted. The if the optional separator (sans brackets) is present, that separator is used to separate multiple values that might be present for a field (i.e., if you have a checkbox field which had multiple values checked). The special case of **\n** is translated to a newline character, if supplied as the separator.

Inclusions and substitutions are fully recursive. If you include a file *someheaders1.html*, which in turn has an include present for a file *somejavascript.js*, which in turn has an include present for yet another file, field, or command, substitutions are performed all the way down the chain. The one case in which this is not true is the field include. If field includes were allowed to recurse, they would open a system up to remote file disclosures and remote code execution injection attacks. File and command inclusions may be recursive because all data is defined server-side. One caveat, however, would be including any files which you know contain fields that include field values which have not been properly sanitised. Since this would require server-side misconfiguration by the developer, it's still allowed to be recursive, but you should be aware of how you utilise includes and make sure that you do not specify risky includes.

It should be noted that when we say field includes are not allowed to recurse, we mean from the point at which the substitution is made. If there is a field include specifed five levels deep within included files or commands, the field will still be substituted. However, the substitution code will not recurse to check for further includes within the field contents, as it does with files or commands.

ENV: variable_name

This directive will include the value of the environment variable specified. If the environment variable is empty or does not exist, a null string will be substituted.

Please note that normal OneGate output substitution tokens (see previous section) *are* available within included files, commands, environment variables, and fields.

Individual Field Substitution

It is possible to substitute individual field values into your output stream, rather than the contents of whole files. In the case of simple values for form fields and other dynamic data, it may be preferable to utilise this functionality with the *sub_fields* option set to **yes**.

The definitions for field substitutions should be written to the file indicated by **%f** or **ONEGATE_FIELDFILE** from somewhere within one of your executed programs. Prior to final output, any fields defined in this file will have their values substituted into place.

The field definitions themselves are arbitrary. The definition of a "field" in this file is entirely up to the developer. If one is used to using filePro, one might use the ***ca** or **!32** syntaxes. If one wishes to do something a bit more readable, one could do something like **&firstname&**, or **#lastname**. The file format is to define a field, followed by whitespace. Everything before the first whitespace is the actual field placeholder/definition. Everything after the whitespace is the value to be substituted into the output stream. Each field/value definition is to be defined one per line.

In this fahion, one could deploy a field substitution file containing:

```
&firstname Mike
&lastname Larson
&address1 P.O. Box 1234
&city Hicksville
&state VA
&zip 12345
```

Using the above example, anywhere **&lastname** appears in the output stream, the value *Larson* would appear in place of that field placeholder. Likewise with the rest of the fields, and their respective values. In the above example, I arbitrarily defined my fields as an ampersand, followed by a keyword. I could just as easily have used filePro-centric ***fn**, ***ln**, ***ad**, etc., type fields, or defined any other format I wanted. As indicated, the field format and definition is *entirely arbitrary*. The format of your field before the first whitespace defines your field placeholder. Additionally, you can mix and match formats of field placeholders, utilising more than one placeholder field format in one file.

There are a few important things to note:

- Field placeholders are *case sensitive*. If you define a field as ***ab**, you *must* use lowercase ***ab** in your output stream to trigger substitution. Using ***AB** will *not* trigger subtitution for that field, in this example.
- Field placeholders may run directly up against other text. If you define a field placeholder **&firstname**, you may run that directly up against both "word" and "non-word" characters and trigger substitution. (e.g., **'&firstname's InBox'**, or **'value="&firstname"'** are both valid.
- Because OneGate was coded to allow running field placeholders directly against other text, it was also coded to *internally* re-sort the field/value definition file, placing similar strings which are longer *ahead* of similar strings which are longer. Thus, you might define ***somefield3**, and ***somefield31**, as well as ***somefield30024**. It doesn't matter in what order you include them in the field file. OneGate will re-sort them so that they are substituted in the order from longest to shortest (30024, 31, and then 3, in this example). In this fashion, you will *not* end up with ***somefield3** triggering a substitution on ***somefield30024** by accident, stranding an extraneous "0024" in the output stream.

OneGate handles the automatically, and you may define all your fields in an arbitrary order within the file, without having to worry about such collisions.

Given that there are three different types of substitutions available as of OneGate v05.01.00, it is worth noting that, assuming all substitutions are enabled, there *is* an order of precedence. The order of precedence is: template substitition, then individual field substitution, and finally output substitution tokens. This order of precedence allows for the greatest flexibility, allowing you to include direct fields and output tokens in files, commands, environment variables, and fields, and allowing you to include internal OneGate tokens within individual fields.

Application Configuration, A Quick Guide

We will now give step-by-step instructions for quickly manually configuring OneGate for use with an application. This assumes that you have OneGate installed, and the global configuration has been performed. The same directory conventions as found in the *Configuration Instructions* section will be used in this section. Bear in mind that you can also use the ***ogconfig*** configuration tool to configure applications, as well as the rest of OneGate. Depending on your preference, this may or may not make things easier/faster.

To configure a new application manually, perform the following steps:

1. Change directory to the *[maindir]/configs* directory.
2. Decide on a program set name, and make a directory with that name.
3. Change directory into your program set directory.
4. Edit the *program.set* file, and enter your desired commands for the program set.
5. Edit the *fields* file, and add all field definitions required for your application.
6. Edit the *cookies* file, and add all cookie names your application needs to have the data retrieved for. This step is optional.
7. Edit the *config* file if local application configuration options are necessary. This step is optional.
8. Edit the *environment* file if you require applications-specific environment variables. This step is optional.
9. Edit the *host.allow* file and any denial message files as necessary if you require application-level virtual firewalling. This step is optional.
10. Edit the *allowed.users* file and create valid *ht.users*, *ht.groups*, and *invalid.user.message* files if you require application-level user authentication. This step is optional.
11. Change directory to *[maindir]*.
12. Edit the *valid.sets* file and add your program set name.

That's it! You should now be ready to test your application, assuming you've configured your application for use with OneGate. The *minimal* configuration required there is that it write the appropriate response file to the *[maindir]/spool/[progset]* directory.

The OneGate Configuration Tool (ogconfig)

The ***ogconfig*** tool can greatly aid you in configuring OneGate more quickly, depending on whether you prefer working manually in the filesystem or with configuration utilities.

This tool is a text-based, menu-driven application that assists you in configuring OneGate. For files that do not exist comments of the relevant type (configuration-style or HTML-style, depending on the file in question) are inserted at the beginning before you are launched into the editor of your choice. These comments detail what should go in the file, and in what format, if the format is specific. These comments should help with the editing process, and reduce the frequency with which you might ordinarily have to check the online documentation. For pre-existing configuration files, no additional comments are added.

Another nice enhancement is the syntax checking. After you finish editing each file, your syntax is checked, and you are notified exactly which line contains any error. If multiple lines contain errors, you are directed towards the first problematic line.

Some things of note about the tool:

- You **must** set the environment variable *EDITOR* to a valid executable in order to use this program. Said editor must be able to take the file to be edited as its first argument. Under *nix systems, just about any editor will meet this criterion, and your environment quite possibly already has this variable set. If not, you will need to set it. For Windows users, you may set this to any editor that meets that prerequisite. I personally prefer ***gvim***, although ***notepad*** works well too.

- For the User Authentication section of the tool, the external program ***htpasswd*** is used, and should be present within your path. If the program cannot be executed, you will receive an error message, but the program will continue onwards. In this event, you are responsible for generating the password file contents manually. The program does come with many web server distributions, so I consider it ubiquitous enough to rely upon its presence.

- Windows users will not be able to use the symbolic link functionality for the User Authentication section of the tool. You will receive errors about not being able to do this, if you attempt it, but the program will continue onwards.

It is our hope that this tool reduces the frustration some people feel over manually configuring applications at the system level. Given a proper environment, you can configure everything about OneGate with this tool with relative ease.

For *nix platforms, log in as or su to the user under which onegate should be running, before using this tool.

The OneGate Cloning Tool (ogclone)

The ***ogclone*** tool can be used to quickly copy entire OneGate program set configurations from one name to another. This is a time-saver when doing mostly "cookie-cutter" style development where most things stay the same between multiple program sets.

Simply run the tool, select a program set to clone, and a name for the new program set. All files from *[maindir]/configs/[progset]* will be copied, as will any files from *[maindir]/spool/[progset]/tmp*, which is the current working directory for each individual program set, and may be used for convenient storage of external application configuration files.

Some things of note about the tool:

- You **must** be either root or the user that owns *[maindir]* on *nix systems in order to run this tool. If you are not, the program will warn you and force you to log in as or su to either.

The OneGate Lockfile Cleaner (ogclean)

The ***ogclean*** tool can be used to clean up any stray lockfiles that correspond to OneGate processes that are no longer running.

Sometimes during development, people generate program sets that hang indefinitely at the command level, and then kill off the jobs and the OneGate sessions. While OneGate traps the common termination signals and cleans its lock files in most instances, signal 9 in *nix (SIGKILL), for example, is entirely untrappable. Using such brute force would result in a stranded lockfile that would be taking up one "slot" in the maximum session count.

This tool is meant to be run either manually or from cron (or a task scheduler in Windows, if needed), and will remove any lockfiles corresponding to processes that are no longer running.

You simply invoke the program by calling *ogclean* with no arguments, and it cleans any stray lockfiles. You will be notified of any lockfiles removed. If you wish to cause the program to always run silently, simply create the file *[maindir]/quietclean* to achieve this effect. That file's presence suppresses the usual non-error output of the lock cleaning program. This file need only be placed in the original installation *[maindir]*, not any testbed versions of *[maindir]*, as all locking is handled from the installation site to make session counts work correctly.

Running OneGate

Running OneGate is quite simple. The most common way CGI programs are invoked is via an HTML FORM, where information is put into fields, and a SUBMIT button is pressed to invoke the CGI application.

Within this methodology, your form's ACTION tag should look like the following:

```
ACTION="http://www.myhost.com/cgi-bin/onegate"
```

In addition, your form must contain one field, which should be of the HIDDEN type. The field should look like the following:

```
<INPUT NAME="onegate_set" TYPE="HIDDEN" VALUE="my_program_set_name"
METHOD="POST">
```

Please note that if you are going to use HTTP File Upload, you *must* add the following to your FORM tag:

```
ENCTYPE="multipart/form-data"
```

Alternately, you may create regular links to OneGate in the following fashion:

```
<A HREF="http://www.myhost.com/cgi-bin/onegate?onegate_set=my_program_set_name">Click
Here!</a>
```

Please note that HTTP File Uploads will *not* work from regular HREF links, and they do require the ENCTYPE attribute, as shown above, as well as the POST method type.

The HREF methodology performs a GET style request rather than POST, and is not recommended for more than a few kilobytes of data. My rule of thumb is to use POST if you have more than 100 characters of data or so. This methodology was meant for *brief* queries, such as those sent to search engines. Sending large amounts of data with the GET method may result in the argument space of

the server's process being exceeded on some platforms, which will corrupt your transactions. This is a technical limitation of the GET method as relates to argument space availability within operating systems. Always use POST for more than the smallest amounts of data, as it also keeps URLs short and manageable, which is considered good form.

An interesting feature to note is that the first time you run an application, it creates its *[maindir]/spool/[progset]* directory. Inside this directory, a *tmp* directory is also created. This directory is persistent. When OneGate runs the application, it changes the *PWD* environment variable to the *[maindir]/spool/[progset]/tmp* directory, and also internally changes directory to this location for the execution of the program set. When your programs run, if they need any persistent configuration files of their own, you can safely store them here and open them from the current directory. Should you need this directory prior to running the application for the first time, simply create the hierarchy leading up to it (*[progset]* and *tmp* itself, starting inside the spool) and make sure your *nix ownership and permissions (or Windows security settings) are set correctly for the directories.

As of OneGate v05.00.00, you can also pass the *onegate_set* parameter as part of the $PATH_INFO segment of the URL, and leave it out of the submitted fields. This was implemented due to the need to integrate with payment gateways and other web services who do postbacks but who do not pass arbitrary fields back to OneGate. The way to do this for POST is:

ACTION="http://www.myhost.com/cgi-bin/onegate/onegate_set=set_name"

The way to do this for GET is:

ACTION="http://www.myhost.com/cgi-bin/onegate/onegate_set=set_name?field1=val&field2=val"

If you supply the *onegate_set* value via the URL, you should *not* include it in your fields. However, the value must be supplied to OneGate via one method or the other.

Multiple Testbed Functionality

OneGate supports using different configurations using the same program set name, so that one may interchangeably use live code and development code at the same point in time This can be useful if you're testing new features that are modular and simply want one module to work in test code, or if you want to use an entire copy of all relevant configurations as a testbed, and alter those configurations to protect your production environment.

By way of example, let us assume that the same command lines, fields, etc., are all required for an entire project, but we don't want to affect live data. Assume that an environment variable triggers the change in which data your application actually utilises (filePro users, consider *PFDIR* by way of example). You can keep your live system running, and also run a development testbed at the same time without having to have multiple servers. The following illustrates the steps required to do so:

1. You will be copying the *onegate* program in your CGI directory to another name. This name is arbitrary. You could call it "onetest", "onealternate", or even something highly cryptic such as "lkjabwlouz". The name doesn't matter, so long as it's a valid filesystem and URL filename. Choose your name at this time, and remember it for the following steps.

2. Copy the CGI program *onegate* to the name you chose, in the same CGI directory. Make sure your ownerships and permissions are correct on the new executable (i.e., cgiwrap is very picky, if you're using that).

3. Change directories to the directory that contains the main *onegate* directory.

4. Copy the entire *onegate* directory tree to a directory of the same name as the CGI program you chose in step 1. (Note to users who install and use their programs with the .pl suffix: The directory name should **not** include the .pl

suffix--only the main name of the program.) Make sure all ownerships are changed to correct values throughout this tree.

To use an alternate testbed, simply call the new CGI name you picked in step 1 above in your URL.

When OneGate sees that it is not called "onegate" or "onegate.pl", it will check for a configuration directory with the same name with which it was invoked, at the same directory depth that the main *onegate* directory resides. (e.g., If your *onegate* directory is */usr/local/lib/onegate*, your directory for a testbed named "onetestbed1" would be */usr/local/lib/onetestbed1*.) If said directory is both present and readable, it will use this directory for **all** configuration values. The one thing to ensure is that you maintain the same *max_session_count* configuration in all testbeds, as OneGate will use the same lock directory for all lockfiles in order to maintain an accurate count of instances, whether one is using one production environment or a production environment and five different testbeds. The *[maindir]/locks* directory (in this specific case, *[maindir]* being what you **installed** into, rather than the testbed version of it) is **always** used for **all** locks, no matter whether you're running production or any one of several testbeds.

You may alter your testbed configurations at will, changing field requirements, configuration options, and all other manner of functionality.

In order to ensure that you can go back through a complete product chain with one testbed or another, it is recommended you utilise the *%p* cookie on the command line, or the *ONEGATE_CGINAME* environment variable. Instead of hardcoding *onegate* into place in a URL (e.g., *http://somehost.com/cgi-bin/onegate*) throughout your application code, you should use the value obtained from the cookie or variable instead. This will let you chain through an entire product using the same production code or testbed as you start with. You may of course use hardwired values for production and simply insert specific testbed names in the URL's for certain modules if you're just adding functionality modularly.

Any testbed whose name ends with **_ajax** will automatically have all default MIME types set to **application/xml** and the **cache** setting defaulted to **no**.

NOTE: The *ogconfig* and *ogclone* tools will work on alternate testbeds if you supply the name of the testbed (the name from step 1 above) as the first argument to either program.

OneGate Logs

OneGate logs its activities in the *[maindir]/logs* directory, and each program set has its own log. The log format is a single line containing the following components:

- The full date and time followed by a colon
- If **new_log_format** is enabled, the session ID for the request
- The IP Address of the remote client that made the request
- The program set name
- The end result of the request. This can be one of the following:
  - SUCCESS
  - LOCK_COUNT_EXCEEDED
  - SUBNET_BANNED

- ACCESS_DENIED
- ERROR_[NUMBER]

5. If the result was an ERROR_#, the rest of the line following a single whitespace comprises the full text of the error type. Additionally, if **new_log_format** is enabled, any diagnostics included by the enabling of **devel_debug** will directly follow the internal error information.

Security Notes

OneGate was designed with security in mind. While it makes every attempt to be a secure application gateway, you have to do your part as well.

The biggest pitfall of CGI programming is trusting external data and using it for system calls without vetting it first. This can be catastrophic, and lead to destruction of data. OneGate does *not* use any remote data in remote commands, nor does it provide you any facility to do so via the program set command file. However, it does not (and can not) prevent you from pursuing this extremely hazardous practice within the applications you call from the command file. Those applications could conceivably call other programs unsafely, if you blindly trust data provided from outside your control and use it in a command line context. **DO NOT DO THIS.** Doing so is tantamount to leaving the keys to your car on the hood while you quickly nip into Wal*Mart for even one tiny item. There's a good chance your car won't be there when you get back. Likewise, bad things will inevitably happen if you commit this grave security error.

OneGate handles upload filenames safely. You are provided a safe filename, generated by OneGate, for use. You are also provided the original filename that the file was known by on the remote system, if the application sends it. However, trusting that remote filename can be *extremely* hazardous. You should always have your software inspect such filenames for rogue content before utilising them.

OneGate simply passes the data it obtains without parsing it or censoring it. As a generic gateway application, it can not and should not do so. It is up to you to make sure that your application adequately scans and validates the incoming data to ensure it is safe for use in whatever context is applicable.

It is your responsibility to make sure that your configurations restrict access to applications as appropriate. For anything not intended for general public consumption, you should implement access controls with the program set virtual firewall and user authentication mechanisms available in OneGate. OneGate can easily cooperate with any HTTP Basic Authentication already in use on your web site.

Security is one of the biggest issues with CGI programming. OneGate does its best to make the environment safe as possible without compromising the data integrity. It is your responsibility to ensure that your applications use the data safely.

For a complete list of points to check over in your integration efforts, please consult this more comprehensive list at http://www.fairlite.com/fc/articles/cgi-security.html that we have released to the public to aid in CGI security education.

Dealing with UID Conflicts

OneGate is run by the http server on your system. It also uses directories and files for various functions. Thus, the directories and files should be accessible to the User ID that httpd runs as, as

well as the applications that it calls. Both OneGate *and* the applications it calls must be able to read and write to these locations.

Many times, this can be achieved by simply making the files and directories owned by the httpd-running user. However, there are circumstances in which you might be calling a program that needs to run SUID as another user.

In cases like this, you have several options:

- If you use Apache, you could use something like the *suEXEC* facilities.
- You can use an application like *CGIWrap*. I use this option myself, and have had no difficulties with it, even under unusual configuration requirements.
- If you administer the system and you are implementing a web server dedicated to one particular task, you can make httpd run as the required user that matches what your application requires.

If you have concerns regarding this issue, you can contact us at **onegate@fairlite.com** and we will assist you in finding the solution that's best for your situation.

OGCGIXML Accessibility for filePro

One application that was highly in mind during the creation of OneGate is filePro. Using OneGate with filePro is quite easy with a filePro *onegate* database and the *OGCGIXML* processing table. This integration package is available with your regular OneGate licensing fee, at no additional charge.

In short, OGCGIXML is a OneGate-specific XML parsing system that takes OneGate-generated XML and populates records in a filePro database, which you may then look up from any of your applications. This makes filePro CGI work just about as "plug and play" as possible.

To install and utilise OGCGIXML, follow the following steps:

1. Extract the OGCGIXML archive for your platform into the appropriate filePro PFDIR directory. The database name that all the relevant files will be placed in is *onegate*.
2. Make sure that permissions are correct for filePro to access the database.
3. In versions of OneGate prior to 04.00.00 with OGCGIXML 1.4 or lower, you had to configure OneGate for your filePro application, passing **-rw %i** on the command line to *[dr](report|clerk)*. In your application, you would look for the value contained in **@PW** in your lookup to the *onegate* table. As of version 04.00.00 *with OGCGIXML 1.5*, all legacy code works correctly as it sits. However, for new applications you can omit the passing via **-rw %i**, and instead look for the value from *ONEGATE_INFILE* when performing your lookup(s). Please bear in mind that it's really the version of OGCGIXML installed that determines whether you can use this less resource-demanding behaviour--that version of the parser was simply released with OneGate 04.00.00.
4. In your processing table, **DECLARE GLOBAL OGParseResult(4,*,g)** so that you may check the parse status after the next step.
5. In your processing table, place a **CALL "onegate/ogcgixml"** statement. The table should only be called *once* to parse the incoming XML data.
6. Check the contents of *OGParseResult*. The potential values are *PASS* on success, or *FAIL* on error.
7. Assuming success, look up your fields in the *onegate* database. The fields are:
    1. Application ID (999,*)
    2. Field Name (100,*)
    3. Field Value (999,*)

4. File Upload Field (1,YESNO)
5. Original Filename (300,*)
6. File Size (20,.0)
7. File Name (300,*)
8. Cookie Field (1,YESNO)
9. TextArea Field (1,YESNO)

The Application ID field should be used for lookups, as all fields for the current transaction have the value contained in the environment variable *ONEGATE_INFILE* stored in that field.

Each record is one name/value relation from the incoming CGI data. If you have multiple values for one name, there will be multiple records, each with the same Field Name, but with the respective Field Values.

Value fields have empty fields for Original Filename, File Size, and File Name, and the File Upload Field will contain *"N"*.

File fields will have an empty Field Value field, contain *"Y"* in the File Upload Field, have populated Original Filename and File Size fields, and contain the safe filename in the File Name field. **You are responsible for handling the actual uploaded file as needed, with your own processing.** (Keep in mind that filePro need not handle the file itself. You *can* use other system commands to deal with the file as needed.)

One *may* utilise the **TEXTAREA** field in the field definition file for a program set to send the contents of a field to a file, rather than to the database itself. filePro users will particularly wish to do this if the anticipated input is greater than 999 characters in length, as this is filePro's real field limitation, and any further contents will be truncated if the **TEXTAREA** option is not used. If you use this method (see the *Configuration Instructions* page), the entire field contents will be saved to a separate file, and the Field Value will contain the path to that file. As with file uploads, you are responsible for handling the TEXTAREA file(s) as necessary.

Fields that are not filled in will not have an entry stored in the *onegate* database. If no entry for a field is present, either the field was optional to OneGate and not submitted, or the field was submitted but left empty. Check accordingly.

In addition, requested cookies are inserted into this database. Cookies will have the Cookie Field set to *"Y"*. In this fashion, it is possible to have the same name appear as both a cookie and as a field in your form, and still tell them apart. Cookies which are requested but not available will not have an entry stored in the *onegate* database. Check accordingly.

Note that you should delete the field records after use. It is not self-cleaning. You can delete them as you go, or you can set up an automated process to clean them periodically to keep the database as small as possible.

Using OGCGIXML is not the only way to access the XML data from filePro. You may supply your own XML parsing routine as you see fit. This integration feature is provided as an added bonus to make web-enabling your applications with OneGate even easier.

Note: Never use the **-bg** or **-bs** flags to *[dr]report* if that is the only program creating the output file. Doing so would cause OneGate to attempt to read the output file, which will not exist yet since filePro would immediately place itself in the background and appear to have completed, when in reality it

would just have started. Even if other programs generate the data, using these flags will more than likely generate anomalous results that you do not want.

For a more complete demonstration of OneGate integration to filePro, please look at my FLFSS project at http://flfss.fairlite.com/ which includes complete maps, source, flow description, and my OneGate configuration for the project.

Note: The following variables are used by OGCGIXML, and should **not** be declared in automatic processing or in a calling table:

- **OGCheck**
- **OGCount**
- **OGDone**
- **OGFHResult**
- **OGFLDebug**
- **OGFMidLine**
- **OGFieldName**
- **OGFieldValue**
- **OGFileHandle**
- **OGFileName**
- **OGInCdata**
- **OGInField**
- **OGInName**
- **OGInVal**
- **OGInputFileName**
- **OGIsCookie**
- **OGIsFile**
- **OGIsTA**
- **OGLine**
- **OGMidLine**
- **OGOMidLine**
- **OGOrigName**
- **OGPMidLine**
- **OGPartCdata**
- **OGTell**
- **OGUpFileSize**
- **OGWholeString**

OneGate and AJAX

OneGate supports AJAX methodology, making it very easy on the server side to let your applications dynamically affect pages without refreshing the entire page. There are a few things to keep in mind when using OneGate for AJAX work. We'll cover them here.

MIME Types

When using OneGate for AJAX work, you should always make sure the MIME type is **application/xml** unless you know what you're doing and have reason to set it otherwise. In the standard paradigm, you're supposed to be returning XML data, so this is the correct setting. You can change it if you need to. But whatever setting you use should be the case across the board. Every MIME type setting for all output types should be set to the same value.

Multiple Testbeds, CGI, and AJAX

Traditionally, OneGate runs in CGI mode. You could set the MIME types all manually for your main "default" testbed personality (i.e., "onegate") if all you were doing was AJAX. However, if you need to use OneGate for multiple roles (such as CGI and AJAX concurrently on the same system), achieving this is quite easy.

In this case, you should use the *Multiple Testbed Functionality* to achieve this. Let us assume you want to use CGI, but you also have some AJAX functionality you'd like to roll in. Your main configurations may all use *text/html*. However, if you create any testbed that ends in **_ajax** (i.e., *onegate_ajax*, *onegate_development_ajax*, etc.), any testbed that ends with that suffix automatically has all MIME types set to **application/xml**. They can be overridden in the configuration files if necessary, but it's an easy way to set yourself up with an AJAX-specific personality for OneGate that lets you coexist with your CGI functionality. Likewise, the default caching is defaulted to off for the AJAX testbeds, but can also be overridden.

Please read the documentation regarding multiple testbeds to learn how to configure these.

Error Checking

OneGate will make use of error files you have written, substituting tokens if you're using that functionality. See: *Output Substitution Tokens* for more information.

However, even if you have a custom error page, there are a few errors that could possibly occur in a misconfigured system prior to the point the error page could be obtained for use. In this case, an internal error format is used.

If this kind of error is triggered, the first thing to note is that the HTTP status code returned from the server will be **244**. This is an extended success code, but it's meant to inform you that you've received OneGate's internal error XML because it had no choice. You should handle this appropriately.

The second thing to know is what you're parsing if you get HTTP status **244** in this type of situation. The DTD for these errors is:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE OG_Error [
```

```
<!ELEMENT OG_Error
(OG_Code,OG_Text,OG_Devel,OG_Version,OG_Program,OG_Session)>
<!ELEMENT OG_Code (#PCDATA)>
<!ELEMENT OG_Text (#PCDATA)>
<!ELEMENT OG_Devel (#PCDATA)>
<!ELEMENT OG_Version (#PCDATA)>
<!ELEMENT OG_Program (#PCDATA)>
<!ELEMENT OG_Session (#PCDATA)>
]>
```

An actual example of the XML would be:

```
<OG_Error>
     <OG_Code>5</OG_Code>
     <OG_Text>No program set given.</OG_Text>
     <OG_Devel>Development debugging disabled.</OG_Devel>
     <OG_Version>04.00.00</OG_Version>
     <OG_Program>Fairlight OneGate v04.00.00</OG_Program>
     <OG_Session>1171416241-7171</OG_Session>
</OG_Error>
```

An example of the JavaScript to get the relevant data would is:

```
if (http_request.status == 200) {
     // Do whatever you'd do on success...
} else if (http_request.status == 244) {
     // OneGate internal failsafe error.
     var xmldoc = http_request.responseXML;
     var ogcode = xmldoc.getElementsByTagName('OG_Code')
[0].childNodes[0].nodeValue;
     var ogtext = xmldoc.getElementsByTagName('OG_Text')
[0].childNodes[0].nodeValue;
     var ogdev = xmldoc.getElementsByTagName('OG_Devel')[0].childNodes[0];
     if (! ogdev) {
          ogdev = 'Development debugging off.';
     } else if (ogdev) {
          var ogdev = xmldoc.getElementsByTagName('OG_Devel')
[0].childNodes[0].nodeValue;
     }
     var ogprog = xmldoc.getElementsByTagName('OG_Program')
[0].childNodes[0].nodeValue;
     // Do something with these values to indicate failure.

} else {
     // Generic transport failure unrelated to OneGate.  Handle apprpriately.
}
```

For anyone doing AJAX work, it's as simple as configuring up your program sets, making sure you add a check for a potential **244** HTTP status code return, and doing the rest of whatever you're doing AJAX-wise. Technically, you don't even need to specifically handle the information that comes back in the case of a **244** request if you only check for status **200** and consider anything else a failure; but if you need the information, it's there.

Change Log

v01.01.09 - 04/30/03

Initial public release.

v01.01.10 - 05/16/03

Fixed issue with application-specific environment variables and double-setting based on bad internal variable name. Had been fixed in global context during development, but not in local context.

---

v01.02.00 - 09/28/03

Changed locking mechanism for master lock file to flock() based locking to avoid race conditions with the old methodology.

---

v01.03.00 - 10/30/03

Added the ability to access cookies.

---

v02.01.00 - 11/29/03

Added subnet and IP deny/allow in program set context. Added user authentication in program set context. Added both global and program set nologin facilities.

---

v02.02.00 - 12/14/03

Created ogconfig configuration tool for easier configuration and less need to reference the documentation.

---

v02.03.00 - 12/28/03

Added 'unlimited' option to max_upload_size option. Added a chdir() to an application-specific temporary directory, as well as the setting of $PWD for children. In this way, external applications used in command lists can find configuration files in an expected place without needing to have things placed in a protected directory.

---

v02.03.01 - 12/30/03

Added *ONEGATE_LOCK_NUMBER* environment variable and the corresponding **%n** magic cookie for command lines. This will be useful with filePro for use with control files where you can start on a specific record number and avoid record-locking deadlocks at simultaneous use.

---

v02.03.02 - 12/31/03

Changed ogconfig to include comments in the command set file skeleton that reflect the magic cookies available to commands.

---

v02.03.03 - 01/01/04

Corrected typo in ogconfig command set comments from v02.03.02.

---

v02.03.04 - 01/08/04

Corrected MIME type validation to include hyphens.

---

v02.04.00 - 02/06/04

Added optional extended TEXTAREA handling, allowing them to be saved to separate files if configured to do so, on a per-field basis.

v02.05.00 - 07/06/04

Added *devel_debug* option and associated functionality.

---

v02.06.00 - 07/16/04

Created ogclone tool. Added *[maindir]/spool/[progset]* and *[maindir]/spool/[progset]/tmp* creation to ogconfig. Released new OGCGIXML parser v1.4 for filePro users.

---

v02.07.00 - 07/27/04

Enhanced devel_debug to show which command in a program set fails proper execution. Added global and local custom error pages that will suppress the normal terse reporting if present (local overrides global). The old error reporting is used if OneGate cannot find a custom error page for the program set in use, or a global custom error page. All custom error pages are disabled if devel_debug is enabled. Added global field and global cookie definition files. Global field definitions will be overridden by any conflicting program set specific field definitions. Updated install.pl to accommodate adding comments to global configuration files if creating them. Updated ogclone and ogconfig to display available program sets in multiple columns. Updated ogconfig with paging menu system, and added all features necessary to handle the new configuration files added in this version.

---

v02.08.00 - 08/17/04

Added multiple testbed functionality, along with the *%p* and *ONEGATE_CGINAME* hooks that make it more readily usable.

---

v02.08.01 - 08/18/04

Modified ogconfig and ogclone to work with multiple testbed functionality. Fixed minor error message bug in ogclone.

---

v02.08.02 - 09/05/04

Fixed install script missing ogclone suffixed variable settings.

---

v02.08.03 - 09/07/04

Fixed user authentication choking with failed authentication if file was present but no users or groups were defined.

---

v02.09.00 - 09/29/04

Added *max_log_size* option and associated functionality. Added full text of errors after error code in log entries. Fixed command-reading internal cosmetic bug.

---

v02.10.00 - 10/08/04

Created *ogclean* tool and modified installer to handle it.

---

v02.11.00 - 10/13/04

Added *%u* and *ONEGATE_UNIQUE_ID*. Made minor tweak to ogconfig to clarify that you can create a new program set, not just configure existing ones.

v03.00.00 - 08/23/05

Added OneBridge hooks.

---

v03.00.01 - 11/13/05

Fixed bug in virtual firewall that still denied access from a host if it matched more than one **allow** rule.

---

v03.00.02 - 09/01/06

Fixed bug in TEXTAREA handling on empty textarea check causing false positives and therefore empty files and path values

---

v03.00.03 - 11/14/06

Implemented *[maindir]/emergency.log* facility for cases where the program may encounter a fatal error before a program set logfile would be selected. Generally speaking this is most often error 5, where the program set name is not specified, although it could be others.

---

v04.00.00 - 02/14/07

Implemented AJAX compatibility, token substitution in output, no_unlink, variable MIME types for all output situations, wildcarded optional fields, reworked OGCGIXML to not need **-rw %i** for filePro, rewrote large segments of the code to optimise for speed, conformed to strict pragma for better debugging, added more user-side debugging help in the form of session tokens combined with no_unlink, and a bunch of other little stuff.

---

v04.01.00 - 07/15/07

Added conditional command execution. Changed default *cache* setting to **no** for *_ajax* testbeds.

---

v04.02.00 - 11/08/07

Fixed wildcard fields matching explicitly declared fields resulting in duplicate XML block entries. Disallowed wildcarded fields being used as FILE or TEXTAREA (now any wildcards are value-only). Enhanced security of file upload filename handling. Added *basename_uploads* option.

---

v05.00.00 - 07/23/12

Added template substitutions, error mails, extended log format, alternate method of passing *onegate_set* inwards, and changed run lockfile format to include timestamp for true uniqueness. Miscellaneous bug fixes and optimisations.

---

v05.00.01 - 08/04/12

Fixed session run lockfile issue which caused the maximum session count to be ignored (related to change in filename structure introduced in v05.00.00). Added email on errors for max session count exceeded, when enabled via general mail on errors functionality

---

v05.00.02 - 08/04/12

Added environment variables to the template include functionality.

---

v05.01.02 – 08/31/12

Added individual field injection substitution mechanism.