<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Component Services</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>1.1 Dependencies</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>1.2 Vulcan Services</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>Configuration Files</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>2.1 Sections</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>2.2 Platform Services</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>2.3 API Services</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>2.4 Platform Options</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>Vulcan Concepts</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td>3.1 The Ming Object Document Mapper (ODM)</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td>3.2 Core Object Classes</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td>3.3 Access Control</td>
<td>24</td>
</tr>
<tr>
<td>4</td>
<td>Extending Vulcan</td>
<td>25</td>
</tr>
<tr>
<td></td>
<td>4.1 Custom Visualizers</td>
<td>25</td>
</tr>
<tr>
<td></td>
<td>4.2 Custom Tools (Applications)</td>
<td>25</td>
</tr>
<tr>
<td>5</td>
<td>License</td>
<td>27</td>
</tr>
</tbody>
</table>

Python Module Index

29
Vulcan is a framework that offers a starting point for collaborative web applications organized around virtual teams. Here we have organized some introductory documentation to familiarize interested developers with setting up a Vulcan deployment and extending it’s capabilities.
Component Services

An application built on the Vulcan framework uses the following component services:

1.1 Dependencies

1.1.1 MongoDB Document Store

MongoDB is used as the primary document store and is primarily accessed through the Ming ODM. This service is horizontally scalable using replication.
1.1.2 SOLR Index

The SOLR index allows for significantly faster lookups and in some cases precaching of views over going straight to the MongoDB database.

1.1.3 Redis Object Store

Redis is used for caching, pub/sub communication, and queueing between services.

This service is horizontally scalable using replication.

1.1.4 Swift/S3 Object Store

Swift, or another S3 API compatible object store, is used for storing and serving files.

1.1.5 SMTP Email Service

An SMTP server must be present to send emails. Typically this is Exim.

1.2 Vulcan Services

1.2.1 ForgeApp

The “ForgeApp” is the TurboGears web application that serves as the HTTP server. Familiarity with TurboGears and WSGI will help when developing and deploying your own ForgeApp. Vulcan offers the ForgeConfig class as an abstract base for your application’s AppConfig.

This is a horizontally scalable service when used with an HTTP load balancer.

1.2.2 Taskd (Task Daemon)

Task (or Task Daemon) is the asynchronous processing service that listens for queued tasks and executes them. Typically one Taskd process is started for each processing core available on the host machine.

This is a horizontally scalable service. Taskd processes line up on a Redis queue for distributing task load. Tasks themselves are functions. When a task is queued the current context (User, Project, etc...) are stored in a vulcanforge.taskd.model.MonQTask object which allows the Taskd process to run the requested task function within the same context and using the same arguments with which it was called.

Declaring a function as an asynchronous task is done with the vulcanforge.taskd.task() function decorator. After decoration, the function can be called directly as a synchronous function and as an asynchronous task with it’s added post method.

vulcanforge.taskd.task(func)

Decorator to add some methods to task functions.

Example:
from vulcanforge.taskd import task

@task
def my_method():
    pass

# synchronous call
my_method()

# asynchronous call
my_method.post()

vulcanforge.taskd.model_task(func)

Decorator to allow ming MappedClass instances to behave as tasks. Functions the same as the task decorator, but called on instance methods.

Example:

class MyMappedClass(MappedClass):
    @model_task
    def my_method(self):
        pass

mc1 = MyMappedClass()
mc1.my_method()  # called synchronously
mc1.my_method.post()  # called asynchronously

1.2.3 Eventd (Event Daemon)

Eventd (or Event Daemon) is synchronous service which registers event types and handlers then triggers those handlers when notified of a matching event. Primarily used by the WebSocketApp to trigger server side processing of client events from WebSocket connections.

This is a horizontally scalable service.

1.2.4 WebSocketApp

The WebSocketApp service offers authenticated WebSocket connections to allow real time bidirectional communication between clients and the application servers. It is a gevent WSGI server.

This is a horizontally scalable service.

1.2. Vulcan Services
Vulcan applications are TurboGears applications and use TurboGears configuration files which are similar to Microsoft Windows INI files and are compatible with the Python standard library’s ConfigParser. The config file specifies service addresses & credentials, platform options, and extensions/overrides.

## 2.1 Sections

Some section inheritance is supported with `use = <PARENT_SECTION_NAME>`.

### 2.1.1 [DEFAULT]

Parameters defined here are inherited in all sections unless explicitly overridden.

- `debug` Boolean stating if debug mode is enabled. **This should always be false in production!** Setting this to `true` will enable an interactive debugging console on errors which is a security vulnerability in a production deployment but a valuable tool during development.

### 2.1.2 [server:main]

This section defines server parameters for running the Vulcan application.

- `host` The IP address to serve the Vulcan application on (0.0.0.0 for all)
- `port` The port to bind to the Vulcan application (80 for HTTP)

### 2.1.3 [app:myapp]

```
use = egg:MyAppClass
```

This section is used by and named for your Vulcan TurboGears application and is where the majority of parameters will be set.

### 2.1.4 [app:myapp_test]

```
# inherit from [app:myapp]
use = myapp
```
This section is used when running tests and is useful to set up alternate service locations and databases for test data.

### 2.1.5 [app:taskd]

```ini
# inherit from [app:myapp]
use = myapp

# set root controller to the task controller
override_root = task
```

### 2.1.6 [app:event]

```ini
# inherit from [app:myapp]
use = myapp

# set root controller to the event controller
override_root = event
```

### 2.1.7 [websocketserver]

This section is used by the `websocketapp` but websocket settings can be set in the `[DEFAULT]` section.

### 2.2 Platform Services

Many of these settings are used in different sections by different services and can be defined in the `[DEFAULT]` section only once instead.

#### 2.2.1 Taskd

- `monq.poll_interval` Interval to poll for new tasks if using polling instead of a Redis queue.
- `task_queue.host` The Redis host to use for the task queue
- `task_queue.port`
- `task_queue.cls vulcanforge.taskd.queue:RedisQueue`

#### 2.2.2 WebSocket

- `websocket.enabled` Boolean stating whether websockets should be enabled for this platform.
- `websocket.host` The IP address to bind the Vulcan Websocket service to
- `websocket.port` The port to bind the Vulcan Websocket service to
- `websocket.auth_api_root` Vulcan app api root. This is the URL where the websocket service will authenticate and authorize with the Vulcan application. Here should be mounted `vulcanforge.websocket.controllers.WebSocketAPIController` or it's subclass.
2.2.3 SMTP/Email

smtp_server  SMTP service host location (i.e. 127.0.0.1 or smtp.example.com)
smtp_port   SMTP service host port
forgemail.host
forgemail.port
forgemail.domain
forgemail.url
forgemail.return_path

2.2.4 Redis

redis.host   Redis service host location
redis.port   Redis service host port

2.2.5 S3/Swift

s3.enabled   Boolean stating whether S3/Swift is available for this platform.
s3.connect_string
s3.password
s3.ip_address
s3.port
s3.ssl
s3.bucket_name
s3.tempurlkey
s3.account_name
s3.app_prefix
s3.prefix
swift.serve_local  Boolean stating if resources should be served through a proxy on the Vulcan deployment. This should only be set true if the S3 server is set up to be not directly accessible to end users.
swift.auth.deployment_id
swift.auth.deployment_param
swift.auth.token
swift.auth.token_param
2.2.6 Mongodb

Credentials and settings for the Mongodb connection through Ming.

  ming.main.uri
  ming.main.database
  ming.main.replica_set
  ming.main.read_preference
  ming.main.autoEnsureIndexes
  ming.project.uri
  ming.project.database
  ming.project.replica_set
  ming.project.read_preference
  ming.project.autoEnsureIndexes

2.2.7 Solr

  solr.host  Solr service location
  solr.port  Solr service port
  solr.vulcan.core Identifier of the vulcan Solr core

2.3 API Services

These are API credentials for web services used in an application.

2.3.1 ReCaptcha

  recaptcha_api_url  www.google.com/recaptcha/api
  recaptcha_public_key
  recaptcha_private_key

2.4 Platform Options

  theme  The theme to use. See themes.

2.4.1 Passwords/User Authentication

  login_lock.engaged  Boolean stating if failed login attempts lock a user account
  login_lock.num  Number of failed attempts that trigger a lock
  login_lock.interval  Duration of time in minutes that failed login attempts count towards an account lock
auth.user.inactivity_period  Time in months, more precisely 30 day periods, after which an account
with no activity is considered inactive and will be disabled with the `paster expire-users
<config>` command

auth.pw.min_length  minimum password length

auth.pw.lifetime.months  Password lifetime after which a user required to change their password

auth.pw.min_lifetime.hours  Span of time in hours after a password change during which the password
cannot be changed again

auth.pw.generations  Number of old passwords that cannot be reused

auth.pw.min_levenshtein  Minimum levenshtein distance allowed between consecutive passwords

idle_logout_enabled  Boolean whether users should be logged out automatically after a period of inac-
tivity

idle_logout_minutes  Span of time in minutes after which a user should be logged out automatically if
idle_logout_enabled is true

idle_logout_countdown_seconds  Span of time in seconds before automatic logout when a user is pre-
sented with a dialog to cancel the automatic logout
3.1 The Ming Object Document Mapper (ODM)

Vulcan applications use the Ming ODM which sits on pymongo to interact with the mongodb database. Most of the persisted core object types in a Vulcan application subclass Ming ming.odm.MappedClass. The Vulcan middleware handles setting up the database session.

For more details see the official Ming Documentation.

3.2 Core Object Classes

Vulcan provides framework support for organizing secured virtual teams. Projects are organized into Neighborhoods
3.2.1 User

The User object does exactly what one might imagine that it does: represents human users of the platform.

```python
class vulcanforge.auth.model.User
    Vulcan’s representation of individual human users.
    SALT_LEN = 8
    active()

    @classmethod
    def active_count()
        Get the total number of active users.
        ‘active’ for now means that a user is not ‘anonymous’, ‘root’ or ‘admin’

        Returns Number of active users
        Return type int
```
add_workspace_tab_for_project (project, with_flash=False)
add_workspace_tabs (tab_descriptors, order=None, safe=True)
address_object (addr)
classmethod anonymous ()
    Get the anonymous user
classmethod by_display_name (name)
classmethod by_email_address (email_address)
    Lookup a User object by an email address.
        Parameters email_address (basestring) – The email address to search for
        Returns A User object or None
classmethod by_id (_id)
classmethod by_username (name)
claim_address (email_address, confirmed=False, is_primary=False)
claim_openid (oid_url)
content_agree_artifact (index_id)
content_agreed_artifacts
delete_account ()
    Disables user and resigns from all projects
delete_workspace_tab_for_project (project)
delete_workspace_tab_to_url (url)
disabled
    Don’t use directly, instead use the get_pref() and set_pref() methods.
display_name
email_address_header ()
email_addresses
    Returns a list of email addresses claimed by this user
expertise
get_email_address ()
    Get this User’s primary email address
        Return type str | unicode
get_pref (pref_name)
get_profile_info ()
get_read_roles ()
get_role_ids ()
    Find all of the User’s ProjectRole IDs
get_roles ()
    Find all of the User’s ProjectRoles
get_swift_cookies
get_swift_params (force_new=False)
get_tool_data (tool, key, default=None)
get_trust_history ()
get_trust_info (force_update=False)
get_workspace_references ()
get_workspace_tabs ()

    Returns JSON string of this User’s workspace tabs (bookmarks)
    Return type str | unicode

has_content_agreed_artifact (index_id)
icon_url (**gravatar_kwargs)
index_dict

    Returns dictionary used for SOLR index entry
index_text_objects
initialize_workspace_tabs (order=None, safe=False)
interests
is_anonymous
    Test if this User is the anonymous user
is_culled (competition=None)
is_real_user ()
kind
    Ming ODM polymorphism field
landing_url ()
    Get the URL this User should be redirected to upon login.
last_login
make_private ()
make_public ()
mission
my_projects ()
    Find the projects for which this user has a named role.
needs_agree_category (term, request=None)
    Tests if this user needs to agree to the Exchange content agreement for the given Exchange Term

    Parameters
    • term – A VehicleForge Exchange Component Term
    • request – (optional)

Todo
remove VehicleFORGE logic from Vulcan class
needs_agree_component \( (\text{component}, \text{request} = \text{None}) \)
Tests if this user needs to agree to the Exchange content agreement for the given Exchange Component

**Parameters**

- **component** – A VehicleForge Exchange Component object
- **request** – (optional)

---

**Todo**

remove VehicleFORGE logic from Vulcan class

---

needs_password_reset

old_password_hashes

open_ids

openid_object \((oid)\)

os_id

password

password_set_at

preferences

private_project()

public

public_key

query = \(<\text{ming.odm.mapper._ClassQuery object at 0x7f3d47b6cf50}>\)

**classmethod register** \((doc, \text{make_project}=\text{True}, \text{neighborhood}=\text{None})\)

Register a new user.

**Returns** instance of \(\text{cls}\) (User or a subclass thereof)

registration_neighborhood()

registration_time

script_name

set_password \((\text{new_password}, \text{as_admin}=\text{False}, \text{set_time}=\text{True})\)

Change this User’s password.

**Parameters**

- **new_password** \((\text{basestring})\) – plaintext password
- **as_admin** \((\text{bool})\) – passed to the authentication provider (for instance as required by LDAP)
- **set_time** – update the password_set_at and needs_password_reset fields of this User

set_pref \((\text{pref_name}, \text{pref_value})\)

set_tool_data \((\text{tool}, **\text{kw})\)

skype_name

state_preferences
store_old_password_hash (hash)
swift_cookie_url (unset_flag=True)
swift_params
tool_data
tool_preferences
trust_cache = None
trust_history = None
trust_info = None
type_s = ‘User’
SOLR indexing object type
classmethod upsert (username)
    Get or create a user object for the given username.

    Parameters
        username (basestring) –

    url ()
    Get a URL for this User.

user_fields
username
validate_citizen
workspace_references
workspace_references_last_mod
workspace_tabs

    Returns
        A Ming (mongodb) query cursor of this user’s workspace tabs (bookmarks)

    Return type
        ming.Cursor

3.2.2 Neighborhood

Neighborhood objects are used to group and organize Project objects.
This organization directly affects the URL of a given project. For instance: A project short-named my-project in the neighborhood with the prefix my-neighborhood will have the URL http://my-platform.something/my-neighborhood/my-project.

class vulcanforge.neighborhood.model.Neighborhood
    Provide a grouping of related projects.

    url_prefix - location of neighborhood (may include scheme and/or host) css - block of CSS text to add to all neighborhood pages
classmethod **by_prefix** *(prefix)*
Prefix without slashes (e.g. projects)

**register_project** *(shortname, user=None, project_name=None, user_project=False, private_project=False, apps=None, tool_options=None, **kw)*
Register a new project in the neighborhood. The given user will become the project’s superuser. If no user is specified, c.user is used.

**user_can_register** *(user=None)*
Whether a user can register on a team project

### 3.2.3 Project

**class vulcanforge.project.model.Project**

**classmethod active_count** *(neighborhood_ids=None)*
Get the total number of active projects.
‘active’ for now means that a project belongs to the ‘Projects’ neighborhood and is not marked as deleted.

- **Parameters** cls –
- **Returns** Number of active projects
- **Return type** int

**add_user** *(user, role_names)*
Convenience method to add member with the given role(s).

**app_config** *(mount_point)*

- **Parameters** mount_point *(str or unicode)* – The mount point of the app
- **Return type** AppConfig

**app_instance** *(mount_point_or_config)*

- **Parameters** mount_point_or_config *(AppConfig or str or unicode)* – The mount point or the AppConfig to lookup
- **Returns** The App or None if there is no matching App
- **Return type** None or App

**classmethod by_id** *(id)*

- **Return type** Project

**first_mount** *(required_access=None)*
Returns the first (toolbar order) mount, or the first mount to which the user has the required access.

**get_expanded_read_roles** *
Returns all roles that have read access

### 3.2. Core Object Classes
get_read_roles()
Returns IDs of most basic role(s) that have read access
For example, if anonymous has read access this method will not return authenticated, because authenticated is a subset
classmethod icon_urls(projects)
Return a dict[project_id] = icon_url, efficiently
classmethod menus(projects)
Return a dict[project_id] = sitemap of sitemaps, efficiently
ordered_mounts()
Returns an array of a projects mounts (tools and sub-projects) in toolbar order.
parent_security_context()
ACL processing should proceed up the project hierarchy.
private_project_of()
If this is a user-project, return the User, else None
users()
Find all the users who have named roles for this project

3.2.4 Applications (a.k.a. Tools)

Users are members of Projects which are containers for Applications, commonly referred to as Tools here for disambiguation, which can be containers for Artifacts.

Vulcan includes several already made tools (Wiki, Tracker, Downloads, Discussions, etc...) along with some functionally required tools (home, neighborhood_home)

A tool is defined in a subclass of vulcanforge.common.app.application.Application and registered with the ToolManager class’s default_tools dictionary. An installed tool for a Project is represented by an

```
vulcanforge.common.app.application.Application
```

 AppConfig instance.

class vulcanforge.common.app.application.Application(project, app_config_object)
Bases: object

The base VulcanForge pluggable application

Variables
- status – the status level of this app. ‘production’ apps are available to all projects
- searchable (bool) – toggle if the search box appears in the left menu
- permissions – a dictionary of named permissions used by the app, the values describe what the permissions enable
- sitemap – a list of SitemapEntries to create an app navigation.
- installable (bool) – toggle if the app can be installed in a project
- self.root (Controller) – the root Controller used for the app
• **self.api_root** (*Controller*) – a Controller used for API access at /rest/<neighborhood>/<project>/<app>/

• **self.admin** (*Controller*) – a Controller used in the admin interface

**admin_menu**()  
Apps may override this to provide additional admin menu items  
:return: a list of SitemapEntries

**classmethod default_options**()  
Returns the default config options

**get_calendar_events**(*date_start*, *date_end*)  
Apps can provide events to the Calendar App

**get_markdown**()  
App definitions can override this method to use a different markdown setup.  

**Returns** an instance of markdown ready to use for conversion  
Implemented to consolidate which markdown logic to the application instance for markdown preview mode. Not used consistently throughout codebase.

**handle_message**(*topic*, *message*)  
Handle incoming email msgs addressed to this tool

**has_access**(*user*, *topic*)  
Whether the user has access to send email to the given topic

**install**(*project*, *acl=None*)  
Whatever logic is required to initially set up a tool

**is_visible_to**(*user*)  
Whether the user can view the app.

**iter_mapped_classes**()  
Iterates yielding each mapped class owned by this application.  
Used under the hood in purging all database objects from a specific app.

**main_menu**()  
Apps should provide their entries to be added to the main nav  
:return: a list of SitemapEntries

**set_acl**(*acl_spec=None*)  
Install default acl. Note that we cannot modify the config acl directly, because ming does not note the change.

**sidebar_menu**()  
Apps should override this to provide their menu  
:return: a list of SitemapEntries

**uninstall**(*project=None*, *project_id=None*)  
Whatever logic is required to tear down a tool

### 3.2.5 AppConfig (installed Tool)

Custom Application subclasses define a tool’s behavior while AppConfig instances represent an installed Application within a Project.

Artifact instances keep a reference to their parent AppConfig which keep a reference to their parent Project.

**class** vulcanforge.project.model.AppConfig  
Configuration information for an instantiated Application in a project
Variables

- **options** – an object on which various options are stored. options.mount_point is the url component for this app instance
- **acl** – a dict that maps permissions (strings) to lists of roles that have the permission. See ACL.

**has_access**(permission, user=None)
Test if a user has a specific permission in this AppConfig.

**Parameters**

- **permission** – The permission to test, (e.g. “read”)
- **user** – The user to test, defaults to c.user

**instantiate**()
Instantiate the Application class for this instance.

**load**()
Load the Application class for this instance.

**parent_security_context**()
ACL processing should terminate at the AppConfig.

### 3.2.6 Artifact

```python
class vulcanforge.artifact.model.ArtifactApiMixin
    Bases: object

    Objects that behave as artifacts within the forge need to implement this

    **get_discussion_thread**(data=None, generate_if_missing=True)
    Return the discussion thread for this artifact (possibly made more specific by the message_data)

    **Return type**
    vulcanforge.discussion.model.Thread

    **has_relations**()

    **Return type** boolean

    **index_id**()
    Globally unique artifact identifier.
    Used for SOLR ID, shortlinks, and maybe elsewhere

    **link_text**()
    The link text that will be used when a shortlink to this artifact is expanded into an `<a>` tag.
    By default this method returns shorthand_id(). Subclasses should override this method to provide more descriptive link text.
```
link_text_short ()
   Shortened version of the link text. Defaults to same as link_text.
   Used for ReferenceBin :return: str

parent_security_context ()
   ACL processing should continue at the AppConfig object. This lets AppConfigs provide a ‘default’ ACL
   for all artifacts in the tool.

primary ()
   If an artifact is a “secondary” artifact (discussion of a ticket, for instance), return the artifact that is the
   “primary”.

raw_url ()
   Url at which to download the resource.
   Returns a url
   Return type basestring

ref_category ()
   Category for grouping types of relations
   Returns the category
   Return type basestring

relations ()
   generator for related artifacts with link context
   (extra field in ArtifactReference.references)

shorthand_id ()
   How to refer to this artifact within the app instance context.
   For a wiki page, it might be the title. For a ticket, it might be the ticket number. For a discussion, it might
   be the message ID. Generally this should have a strong correlation to the URL.

url ()
   Subclasses should implement this, providing the URL to the artifact

class vulcanforge.artifact.model.Artifact
   Bases: vulcanforge.common.model.base.BaseMappedClass, vulcanforge.artifact.model.ArtifactApiMixin

The base class for anything you want to keep track of.

It will automatically be added to solr (see index() method). It also gains a discussion thread and can have files
attached to it.

Variables
   • tool_version – default’s to the app’s version
   • acl – dict of permission name => [roles]
   • labels – list of plain old strings

get_link_content ()
   Text content that contains the artifacts shortlinks
   Used for parsing references

get_read_roles ()
   It can only be more restrictive than the project read roles
**Returns** list of role IDs with read permission

**Return type** list

`index()`  
_text_objects=None, use_posts=True, **kwargs

Subclasses should override this, providing a dictionary of solr_field => value. These fields & values will be stored by solr. Subclasses should call the this and extend it with more fields. All these fields will be included in the 'text' field (done by search.solarize())

The _s and _t suffixes, for example, follow solr dynamic field naming pattern. You probably want to override at least title_s and text to have meaningful search results and email senders.

`index_parent()`  

For artifacts whose parent indexes should be updated when they are updated (e.g. Posts)

**Returns** the parent artifact if it exists

**Return type** Artifact | None

### 3.3 Access Control

Access control is defined by the use of Access Control Lists (ACLs) and Access Control Entries (ACEs). *Neighborhoods, Projects, AppConfigs, and Artifacts* each have an ACL which inherits entries from it's parent.
A Vulcan application’s capabilities can be extended in several ways which allow solutions to be custom tailored for a broad range of domains.

4.1 Custom Visualizers

Registering custom visualizers capable of rendering proprietary file formats or adding new views beyond highlighted source text is remarkably easy with the provided Visualizer hooks.

4.1.1 Basic Rendering Visualizer

A simple rendering visualizer can be added via the web UI in the designated forgeadmin project (specified in the config file.) A visualizer package is a ZIP file containing at least an html entry point and a `manifest.json` which declares the entry point.

When triggered the designated entry point (i.e. - `index.html`) will be loaded with the GET query parameter `resource_url` being the URL of the file to be visualized. The entry point html is responsible for using javascript or some other means of retrieving the resource and displaying whatever it is designed to display.

4.1.2 Basic Processing Visualizer

Processing Visualizers further extend the flexibility of the Visualizer system with a preprocessing step. Examples of this feature are calculating metrics out of a large proprietary file format into an easily parseable JSON representation or processing parametric CAD files into compact tessallated formats to be loaded into interactive previews using WebGL.

4.2 Custom Tools (Applications)

*Applications* can be seen as tools available to teams (*Projects*) and are sometimes referred to as Tools for disambiguation. Adding specific functionality to users within the scope of their project is done by adding new Tools to the platform. The Tool system is designed to be extended easily.

Adding a new Tool is done by defining your Tool as a subclass of `Application` and registering it as a tool with your Vulcan deployment’s `ToolManager` class’s `default_tools` dictionary.
Copyright (c) 2011, Vanderbilt University. Developed with the sponsorship of the Defense Advanced Research Projects Agency (DARPA).

Permission is hereby granted, free of charge, to any person obtaining a copy of this data, including any software or models in source or binary form, as well as any drawings, specifications, and documentation (collectively "the Data"), to deal in the Data without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Data, and to permit persons to whom the Data is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Data.

THE DATA IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS, SPONSORS, DEVELOPERS, CONTRIBUTORS, OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE DATA OR THE USE OR OTHER DEALINGS IN THE DATA.
V
vulcanforge, 22
vulcanforge.artifact, 22
vulcanforge.common.app, 20
vulcanforge.common.app.application, 20
vulcanforge.neighborhood, 18
vulcanforge.project, 19
vulcanforge.taskd, 4