KosmicTask

The Secure Integrated Scripting Environment for OS X

Guide

Updated March 20

Overview

KosmicTask is a Secure Integrated Scripting Environment for OS X.

KosmicTask can create, edit, execute and share script based tasks using a wide range of scripting languages.

KosmicTask is available as a time limited trial. After the trial version expires you will be able to read but not execute any tasks that you have created until a full licence is purchased and installed.

System Requirements

- Mac OS X 10.6
- 50 MB free hard disk space

Getting Support

Their are a number of ways to request support for KosmicTask.

Application. From the application menu select **KosmicTask - Feedback and Support...**

Forum. Visit the the support forum at forum.mugginsoft.com

Website. Visit the support website at www.mugginsoft.com/kosmictask/support

Email. Send an email request to support@mugginsoft.com

Task Based Scripting

A Task contains a script and a list of the inputs that it requires. When a task is executed the user is queried for the task inputs, the task is executed and the results are returned to the user.

Tasks may be published, enabling them to be accessed and executed by any other network connected KosmicTask user. In addition, users that are able to authenticate over the network will be able to create, edit and execute tasks on the remote machine.

Quick Guide to Running Tasks

To run a task simply select it from the left hand sidebar. Enter any inputs that are required and click either the large green run button or the run button the menu bar. Task results will be displayed in the adjoining pane. To save the task results select **File** - **Save As**… from the menu.

Initially only tasks available to all users will be displayed (this is is the **Public** task view). To access all tasks on the machine select the **User** task view above the sidebar.

To run tasks on another machine simply select the machine name and the required task from the sidebar. To access user tasks on the remote machine you will be asked to authenticate. Enter a valid OS X username and password. By default all exchanges with remote machines are secure.

Quick Guide to Creating Tasks

To create a task select the **Admin** task view and click **New** on the toolbar.
Then, in the displayed **Edit** window:

1. Enter task name, script type, group and description in the first pane.
2. If the Task requires inputs click + to add. Enter the input name and select its type.
3. Click Edit button in the toolbar and select a template from the displayed list. Edit the task.
4. Build the task if required (it is optional for some task types).
5. Click the Run button to display the test pane.
6. Run and test the task.
7. Close the Edit window and click Save to save the task.

Quick Usage Guide

KosmicTask can be used in a wide range of different situations and has features that appeal to various classes of users.

Home. Use KosmicTask on a single Mac or home network to carry out repetitive administration and automation tasks. Non technical users can easily utilise existing tasks. More technical users will find that there is a wealth of scripts available on the Internet that they can easily adapt to their purpose.

Business. KosmicTask is an effective tool for sharing business automation tasks with end users. Document processing, file processing and automation based services can all be configured from within the application and shared out to clients on the network.

Education. KosmicTask provides a great way of introducing students to a wide range of scripting languages. The application comes with a range of both native OS X and embedded scripting languages. Students can develop and test scripts on own machine whilst educators can review and appraise their tasks via the network.

System Administration. Systems and network administrators will find that the application is a convenient and secure method of providing user scripts and performing script based network maintenance.

Software Development. Developers can use KosmicTask to quickly prototype code ideas. It is also useful for building and organizing a library of executable code samples and snippets.

Science and Technology. The distributed task model means that the application is ideally suited to data acquisition and processing projects. Tasks can be configured to collect and process data on the remote target before returning the results for further analysis and persistence.

Application Architecture

KosmicTask is implemented using a multi tier architecture:

Tier 1: KosmicTask Client User Interface
Tier 2: KosmicTask Server
Tier 3: KosmicTask Task Runner
Tier 4: Script Language Executable

The interaction between the tier is as follows;

1. Users interact with the application at **Tier 1** to create, edit and execute tasks.
2. The server in **Tier 2** receives requests from clients on **Tier 1** (both local and remote) to create and execute tasks.
3. Each supported scripting language has an associated Task Runner. When a task is executed the appropriate TaskRunner is launched along with a copy of the script to be executed. This behaviour ensures that each script runs in its own process in **Tier 3** and cannot bring down the server in **Tier 2**.
4. KosmicTask supports both in and out of process scripting. Those scripts that run in process (such as AppleScript and the Cocoa bridge components detailed below) are loaded and executed with the Task

Runner in **Tier 3**. Scripts that are run out of process, such as the majority of the command line derived scripting components, are executed in **Tier 4** as instances of individual scripting subsystems.

When a task execution is requested information is passed down through the tiers, the task is executed in the relevant task runner, and the task results are then returned to the user.

KosmicTask can be configured to load just the server component when a user logs in so that other network users always have access.

Application Tasks and User Tasks

KosmicTask comes preinstalled with a number of general utility tasks known as **Application Task**s. The majority of these are written in AppleScript. **Application Tasks** cannot be edited or deleted but they can be duplicated.

Tasks created by the user are identified as **User Tasks**. These can be created, edited and deleted by the user.

Resource Browser

In KosmicTask each scripting language is implemented as a separate plug-in bundle. The application **Resource Browser** provides access to the resources supplied by each language plug-in. These resources include:

- A number of task templates (see below).
- A number of informational documents.
- A default list of property settings.

The informational documents generally provide links to web based sources of information for the represented language. A usage document provides an overview of how to accomplish basic tasks using the language. This document is intended to accompany and expand upon the information contained with the individual templates.

Each language also defines a default set of property settings. Most of these are read only and define the specific properties of the language plug-in. Several read-write properties also exist, such as the path to OS X scripting subsystems and task function and class names. These read-write properties can be overridden for individual templates and tasks. Selecting a property from the property list will display a short explanation of its purpose below the list.

Template Support

In order to aid user task creation a number of task templates are available through the application Resource Browser. These templates serve to indicate the basic syntax required in each scripting language to achieve a particular objective. In general, all of the distinctive features of KosmicTask are available as a template in each scripting language. Each template includes the script code, the settings necessary to configure the script and an informational document.

Often the best way to find out how to accomplish a particular task is to examine the available templates. The simplest template - Hello Kosmos - simple returns Hello Kosmos! to the user. Other templates indicate how to return complex objects and files as results while others detail features such as class creation and application automation.

In addition to the templates that are supplied with the application (the Application Templates) the user may define any number of additional templates.

Task Inputs

KosmicTask provides a number of plugin based inputs that provide an effective means of acquiring the input to a task. Any number and selection of input types can be defined to provide the parameters for a given task.

The standard input types are:

- Date
- File Contents
- List Item
- Number
- Text

Task Results

When a task completes it may generate a result that is returned to the user. Task results may consist of simple object such as text string or a number. More complex results containing data such as arrays and dictionaries (aka associative arrays or hashes or maps) can also be returned.

Complex results also permit the returning of the contents of a file or files as all or part of a result. Complex results retain their structure when presented to the user and thus can be navigated using an outline controller. More importantly, the results can be saved as external XML files or property lists that maintain the result content and structure.

In addition, CSS based style commands can be embedded in complex results in order to provide result styling.

Complex Task Results

Complex task results are generally returned as YAML formatted strings. YAML (YAML Ain't Markup Language or Yet Another Markup Language) is a straight forward human readable way of representing complex data types containing strings, scalars, arrays and dictionaries. Its simple syntax makes it easy for traditional command line scripting languages like Bash to generate complex results simply by printing YAML formatted strings to their standard output.

YAML is a popular method of representing data relationships and bindings exist for many scripting languages. KosmicTask makes use of these bindings for Java, JavaScript, Perl, Php, Python and Ruby within application defined controller objects. These controller objects handle the translation between the scripts native data structure and YAML. Thus the task produces its result as a native data structure and then passes it over to the controller for formatting. This YAML formatted result is then returned to the user.

See below for the even simpler way in which Cocoa based tasks can return complex results.

Application Defined Scripting Resources

A scripting language plugin can define additional resources to assist with script building and task control.

Most scripting languages provide some means of validating a script prior to its execution. This functionality is invoked by KosmicTask whenever a task build is requested. Some languages lack this integrated facility and rely on additional tools to provide external validation. A language plugin may optionally include such a tool and invoke it during the build stage.

Scripts that run in-process with their task runner normally have access to a KosmicTask supplied controller object. This object is designed to allow the task to interface with the task runner. The controller is generally an object written in the target scripting language. In some cases the task will have to explicitly create a reference to or an instance of the controller object. In other cases the controller object will be pre -defined with the tasks scope.

A language plugin can also provide access to additional scripting facilities. This is the case with appscript which is supplied with the Ruby and Python plug-ins to provide an additional automation interface.

Task File Handling

KosmicTask permits the contents of any number of files to be passed as inputs and/or returned as results. File contents are passed as inputs using the File Contents input type. When the task is executed the file contents are transferred to the target machine and temporarily stored. The path to this temporary file as passed as the actual task input parameter enabling the task to access the file data.

File contents are returned as results by returning a dictionary type result that contains a kosmicFile key defining the file or files whose contents are to be returned to the user.

Temporary File Handling

A task may return the contents of a pre existing file as a result or it may wish to return the contents of a temporary file that has been generated by the task. To assist with this KosmicTask provides a number of ways of requesting temporary file storage. File storage allocated in these way will only exist for the lifetime of the task and will be deleted from the host machine when the task terminates.

A path to a temporary file or files may be requested in three different ways dependent on the nature of the scripting language plugin.

1. Command line based scripting languages simply need to create files in the task's initial working directory.
2. Most in process tasks can call the resultFileWithName function on a KosmicTask supplier controller object.

3. Scripts which send Apple events can send the "result file with name <basename>" command to the KosmicTask application object.

The easiest way to determine what method to use for a particular scripting language is to examine the 'file' or 'files' template or read the usage document in the Resource Browser.

Scripting Language Support

KosmicTask uses an extensible plug-in architecture to support a wide range of scripting languages. By design support is included for all the scripting languages that are present as part of a standard OS X 10.6 installation. When such a task is executed its script is passed to the local scripting subsystem for execution.

In addition, KosmicTask includes a number of additional scripting languages. Support for these languages is wholly embedded in the application bundle and nothing is installed outside of the application itself.

KosmicTask supports both interpreted scripting languages (such as Php and Perl) and those that are compiled (such as AppleScript and Java).

Support is also included for a number of Cocoa bridges. These bridges are extremely powerful and enable scripts to access all the power and functionality of the OS X Cocoa framework.

KosmicTask supports the scripting of tasks in the following languages:

AppleScript
Bash shell
C shell
C (embedded)
C++ (embedded)

Java
Javascript
Korn shell
Lua (embedded)
Perl
PHP
Python
Ruby
Tcl
Tenex C shell
Z shell

(C and C++ interpreted scripting support is provided by CINT)

KosmicTask supports Cocoa framework scripting using the following bridges and components:

AppleScriptObjC	AppleScript bridge
F-Script (embedded)	F-Script Cocoa scripting environment
JSCocoa (embedded)	JavaScript bridge
LuaCocoa (embedded)	Lua bridge
PyObjC	Python bridge
RubyCocoa	Ruby bridge

Within KosmicTask these components are represented by the following standardised names (in the order as listed above):

AppleScript Cocoa
F-Script Cocoa
JavaScript Cocoa

Lua Cocoa
Python Cocoa
Ruby Cocoa

Cocoa Based Tasks

Cocoa based tasks have a great advantage when it comes to generating task results in that they can returned native data structures. KosmicTask will receive the result via the bridge and return it to the user.

Thus, Cocoa based tasks can generate complex results using the relevant scripting language native data structures such as arrays and dictionaries. Native dictionaries can be used to provide a fast and efficient way of returning the contents of files as results.

Cocoa Based Application Tasks

Tasks which utilise the Cocoa framework are run as fully fledged Cocoa applications.

Many Cocoa components require the existence of a run loop in order to be utilised to their full potential. Normally, application tasks exit when the function that launches them exists. Cocoa based tasks, however, can request that they be allowed to continue running in order to receive further input from runloop sources. When the task determines that it is complete it terminates itself and the task results are returned to the user.

Comparing Standard and Cocoa Based Language Implementations

The differences between the standard and Cocoa variants of any scripting language can be summarised as follows:

1. Most standard scripting components run as external command line based process (though the likes of JavaScript does not).
2. The standard scripting components are less complex and will load faster than the Cocoa variants.
3. The Cocoa based scripting components run in process with their task runner.
4. The Cocoa based scripting components will generally use more system resources than the standard components.

We can take the examples of Ruby and RubyCocoa to illustrate the above points.

When a standard Ruby task is executed an instance of the KosmicTaskRubyRunner program is launched (this will be visible in the Activity Monitor application). This program receives the task request from the KosmicTask server component and launches a separate instance of the Ruby command line system (also visible in the Activity Monitor). The task runner passes the task inputs to Ruby and waits for the task results to be returned. When the Ruby instance terminates the task runner formats the task results and returns them to the server component which then transfers them via the network to the user.

When a RubyCocoa task is executed an instance of the KosmicTaskRubyCocoaRunner program is launched. This program executes as a fully fledged Cocoa application complete with runloop. Once successfully launched the task runner loads the Cocoa task into its own process space and executes the designated run function on the designated run class. When the initial run function returns the task runner checks the status of the KosmicTaskController keepTaskAlive property. If the task requests that it be kept alive then the task is allowed to continue executing until it calls the KosmicTaskController stopTask: method. If the task does not request to be kept alive then the task run loop is stopped immediately.

Automation Support and System-wide Scripting

OS X provides a powerful mechanism for automating and controlling system resources and applications using Apple events. AppleScript has been the traditional means of achieving this but now there are other powerful and more intuitive alternatives.

OS X includes a technology called Scripting Bridge which allows Cocoa based applications to automate other services by sending Apple events. All of the Cocoa capable scripting components supported by KosmicTask can utilise this technology to achieve automation. Scripting Bridge therefore means that automation can be achieved by a much wider range of scripting languages than was previously possible.

But there is another alternative…

Apple events are complex and the desired interaction between the controlling script and the targeted application or service can sometimes be difficult to achieve. The appscript component, which is available to the Python and Ruby scripting components, provides an alternative means of generating, sending and interpreting Apple events.

Security

KosmicTask uses Transport Layer Security (the successor to SSL) to provide secure network communication. The OS X keychain is used for secure password storage and retrieval.