Kompira Documentation

Release 1.6.1

Kompira development team

Sep 30, 2020

CONTENTS

CHAPTER

ONE

ADMINISTRATION GUIDE

Author Kompira development team

1.1 Introduction

This manual contains useful information to help the user know how best to manage Kompira.

Please refer to this manual to learn about installation, updates, overall Kompira settings and logs etc.

In this manual, we use `$` for general user and `#` for root user on Linux command prompt.

```
$ echo 'command execution by general user'
# echo 'command execution by root privileged user'
```

1.2 Kompira package management

This manual will explain the installation and updates of Kompira related packages.

See also:

In this section, only the circumstances where the Kompira package is operated on a single server are explained. Please refer to *High Availability (HA) Management* for details on how to run Kompira in a redundant configuration.

1.2.1 Type of installation package

Kompira has the following types of packages.

Package name	Description
Kompira Package	Packages containing Kompira itself. Including: Kompira core function group, job manager and event transmission script.
Job manager package	Packages that includes a job manager and an event transmission script.
Send-Event package	Packages including Send-Event scripts

If you are using Kompira for the first time, please first install the Kompira package.

Use the job manager package when you want to start the job manager process, in addition to the server on which the Kompira package is installed.

The send-event package is used when you want to send an event to Kompira from another server. For integration between systems using event sending, please refer to *Coordination with other systems* .

1.2.2 Install Script

By using install.sh, you can install Kompira's various packages.

```
install.sh [options]
```

The installation process includes installation of middleware to be used by Kompira, construction of databases and automatic startup of processes.

install.sh creates a log file named *install.<process number>.log* regardless of the success or failure of the command.

Note: Before installing on Red Hat, you need to subscribe in advance.

Limitation

install.sh is supported only for RHEL / CentOS installation.

Install.sh downloads various middleware used by Kompira. Please run it when you are able to connect to the Internet.

When connecting to the Internet via a proxy, run install.sh with the `--proxy` option as follows:

```
# ./install.sh --proxy proxy:3128
```

Note: Please set "proxy" and "3128" as the proxy server's host name (or IP address) and port number.

For a proxy server with authentication, run install.sh with "user" as the user name and "password" as the password as shown below.

```
# ./install.sh --proxy user:password@proxy:3128
```

Command Line Options

The options that can be specified for install.sh are as follows.

Options	Description
`--https`	Restricts access to only HTTPS to the Kompira server (default from Kompira v1.5.0). When accessed with HTTP, it will automatically be redirected to HTTPS.
`--no-https`	Allows HTTP access to the Kompira server.
`--no-backup`	Skips the database backup and restore processes.
`--initdata`	Explicitly initialize the database.
`--initfile`	Explicitly initialize the storage destination of the attached file.
`--force`	Even if the major version of Kompira is different, it will force the installation. Also, it will delete the database without confirmation when there is an existing database, initialize the database and attempt installation.
`--proxy <proxy>`	Specify the URL of the proxy server and install. The proxy server specified here is set as the environment variable of the Kompira service, and it is also applied when accessing external HTTP from the job flow.
`--temp--proxy <proxy>`	Specify the URL of the proxy server to be applied only during installation and install.
`--locale-lang <LANG>`	Specify the locale and install.
`--locale-timezone <ZONENAME>`	Specify the time zone and install.
`--jobmngr <kompira_ip>`	Install and update the Job Manager package. It is necessary to specify the host name or IP address of the server on which the Kompira package is installed.
`--sendevt <kompira_ip>`	This will install and update the send-event package. It is necessary to specify the host name or IP address of the server on which the Kompira package is installed.

The jobmngr and sendevt options are exclusive.

1.2.3 Kompira Package

How to install and update the Kompira package itself.

Installation

Extract the Kompira package and run install.sh. Replace <version> with the version number of Kompira.

```
$ tar zxf kompira-<version>-bin.tar.gz
$ cd kompira-<version>-bin
# ./install.sh
[2020-09-17 02:00:24] ****:␣
↪************************************************************
[2020-09-17 02:00:24] ****: Kompira-1.6.0:
[2020-09-17 02:00:24] ****: Start: Install the Kompira
[2020-09-17 02:00:24] ****:
[2020-09-17 02:00:24] INFO:     SYSTEM                  = CENT
[2020-09-17 02:00:24] INFO:     SYSTEM_NAME             = cent8
[2020-09-17 02:00:24] INFO:     SYSTEM_RELEASE          = CentOS Linux release 8.2.
↪2004 (Core)
[2020-09-17 02:00:24] INFO:     SYSTEM_RELEASEVER       = 8.2.2004
[2020-09-17 02:00:24] INFO:     PLATFORM_PYTHON         = /usr/libexec/platform-
↪python
[2020-09-17 02:00:24] INFO:     PYTHON                  = /bin/python3.6

...
```

(continues on next page)

(continued from previous page)

```
[2020-09-17 02:02:46] ****: ------------------------------------------------------------
↪------
[2020-09-17 02:02:46] ****: Test access to kompira.
[2020-09-17 02:02:46] ****:
[2020-09-17 02:02:48] INFO: Access succeeded:   <div class="brand-version">1.6.0</div>
[2020-09-17 02:02:48] ****:
[2020-09-17 02:02:48] ****: Finish: Install the Kompira (status=0)
[2020-09-17 02:02:48] ****:␣
↪************************************************************
```

The installer will automatically install the Kompira package. If "Finish: Install the Kompira (status=0)" is displayed, the installation has been a success.

When installation is completed, please access the Kompira server from a Web browser with the following URL and confirm that the login screen is displayed.

At this time, a warning about the server certificate is displayed. To prevent this warning, please install the SSL certificate on Apache on the Kompira server.

```
https://<Hostname or ipaddress of Kompira server>/
```

Note: To access by HTTP, you will need to install it with the `--no-https` option.

For details on how to operate Kompira with a Web browser, see the operation guide manual.

Update

How to update the Kompira package when it is already installed:

Kompira's version number format is specified as follows.

```
1.<major-version>.<minor-version>
```

Updates where only minor version numbers are changed are called minor updates, and updates where major version numbers are changed are called major updates.

For example, updating from version 1.5.0 to 1.5.2 is a minor update, updating from version 1.4.10 to 1.5.0 is a major update.

Major updates are updates that may contain changes in architecture configuration and DB schema definition, so a different process may be required.

Please check the current version and the Kompira version that you are updating.

Minor update

For minor updates, run install.sh without options.

```
$ tar zxf kompira-<version>-bin.tar.gz
$ cd kompira-<version>-bin
# ./install.sh
[2020-09-17 22:56:32] ****:␣
↪************************************************************
[2020-09-17 22:56:32] ****: Kompira-1.6.0:
[2020-09-17 22:56:32] ****: Start: Install the Kompira
```

(continues on next page)

(continued from previous page)

```
[2020-09-17 22:56:32] ****:
[2020-09-17 22:56:32] INFO:     SYSTEM                    = CENT
[2020-09-17 22:56:32] INFO:     SYSTEM_NAME               = cent8
[2020-09-17 22:56:32] INFO:     SYSTEM_RELEASE            = CentOS Linux␣
↪release 8.2.2004 (Core)
[2020-09-17 22:56:33] INFO:     SYSTEM_RELEASEVER         = 8.2.2004
[2020-09-17 22:56:33] INFO:     PLATFORM_PYTHON           = /usr/libexec/
↪platform-python
[2020-09-17 22:56:33] INFO:     PYTHON                    = /bin/python3.6

...

[2020-09-17 22:57:00] ****: ------------------------------------------------
↪---------------
[2020-09-17 22:57:00] ****: Check version of Kompira installed.
[2020-09-17 22:57:00] ****:
[2020-09-17 22:57:00] INFO: VERSION=1.6.0b4 [pip=/opt/kompira/bin/pip]
[2020-09-17 22:57:00] INFO: A compatible version is installed.

...

[2020-09-17 22:58:18] ****: ------------------------------------------------
↪---------------
[2020-09-17 22:58:18] ****: Test access to kompira.
[2020-09-17 22:58:18] ****:
[2020-09-17 22:58:19] INFO: Access succeeded:   <div class="brand-version">1.
↪6.0</div>
[2020-09-17 22:58:19] ****:
[2020-09-17 22:58:19] ****: Finish: Install the Kompira (status=0)
[2020-09-17 22:58:19] ****:␣
↪************************************************************
```

The installer will automatically update the Kompira package. If “Finish: Install the Kompira (status=0)” is displayed, it has been successfully installed.

When the update is completed, please log in to Kompira from a web browser and confirm that the version number of Kompira has been updated.

Major update

In the case of major update, update it using the following procedure.

- Use the export_data command to retrieve data from Kompira
- Install Kompira in database initialization mode with the `--initdata` option with the install.sh command
- Save the first data extracted to Kompira with the import_data command

Please note that the existing database will be initialized when you run the install.sh command.

```
$ cd kompira-<version>-bin
$ /opt/kompira/bin/manage.py export_data --owner-mode --virtual-mode / >␣
↪backup.json
# ./install.sh --force --initdata
$ /opt/kompira/bin/manage.py import_data --owner-mode --overwrite-mode␣
↪backup.json
[2018-04-09 21:44:15,936:30953:MainProcess] INFO: import data: start...
[2018-04-09 21:44:16,010:30953:MainProcess] INFO: import object: imported
↪"system/types/TypeObject" to "/system/types/TypeObject" (updated)
```

(continues on next page)

(continued from previous page)

```
[2018-04-09 21:44:16,022:30953:MainProcess] INFO: import object: imported
↪"system/types/Directory" to "/system/types/Directory" (updated)
[2018-04-09 21:44:16,033:30953:MainProcess] INFO: import object: imported
↪"system" to "/system" (updated)

...

[2018-04-09 21:44:22,126:30953:MainProcess] INFO: import fields: /user/data:␣
↪[]
[2018-04-09 21:44:22,164:30953:MainProcess] INFO: import fields: /user/data/
↪nodes: []
[2018-04-09 21:44:22,202:30953:MainProcess] INFO: import fields: /user/data/
↪accounts: []
[2018-04-09 21:44:22,218:30953:MainProcess] INFO: import data: finished␣
↪(created=0, updated=59, skipped=0, error=0, warning=0)
```

When running install.sh, specify the `--initdata` and `--force` options to initialize the database.

When the import_data process is completed, please log in to Kompira from a Web browser, confirm that the version number of Kompira has been updated, and that previously created Kompira objects still exists.

Note: When updating from Kompira to Ver. 1.6 from Ver. 1.5 or earlier, full compatibility is not guaranteed and the migrated job flow and library objects may not work as they are. If necessary, check the operation of each Jobflow or Library object after modifying them.

1.2.4 Job manager package

Explanation of how to conduct an installation/package update including for job manager and send-event script.

Installation

Extract the Kompira package and run install.sh. Since the job manager communicates with the Kompira server, you will need to specify the host name or IP address of the server on which the Kompira package is installed as an argument to install.sh.

Replace <version> with the version number of Kompira.

<kompira_ip> is the host name or IP address of the Kompira server.

```
$ tar zxf kompira-<version>-bin.tar.gz
$ cd kompira-<version>-bin
# ./install.sh --jobmngr <kompira_ip>
[2020-09-18 00:42:54] ****:␣
↪************************************************************
[2020-09-18 00:42:54] ****: Kompira-1.6.0:
[2020-09-18 00:42:54] ****: Start: Install the Kompira
[2020-09-18 00:42:54] ****:
[2020-09-18 00:42:54] INFO:     SYSTEM                  = CENT
[2020-09-18 00:42:54] INFO:     SYSTEM_NAME             = cent8
[2020-09-18 00:42:54] INFO:     SYSTEM_RELEASE          = CentOS Linux release 8.2.
↪2004 (Core)
[2020-09-18 00:42:54] INFO:     SYSTEM_RELEASEVER       = 8.2.2004
[2020-09-18 00:42:54] INFO:     PLATFORM_PYTHON         = /usr/libexec/platform-
↪python
```

(continues on next page)

(continued from previous page)

```
...

[2020-09-18 00:43:24] ****: ------------------------------------------------------------
↪------
[2020-09-18 00:43:24] ****: Setup kompira-jobmngrd.
[2020-09-18 00:43:24] ****:

...

[2020-09-18 00:43:24] VERBOSE: run: systemctl restart kompira_jobmngrd
[2020-09-18 00:43:24] ****:
[2020-09-18 00:43:24] ****: Finish: Install the Kompira (status=0)
[2020-09-18 00:43:24] ****:␣
↪************************************************************
```

The installer will automatically install a new job manager package. If "Finish: Install the Kompira (status=0)" is displayed, the installation has been a success.

For details on how to check that the job manager process is running correctly , see *Starting / stopping the Kompira daemon and Checking the status.*

Also, you can check whether Kompira itself is communicating with the job manager correctly from Kompira's "Management area setting page". Please log in to Kompira with Web browser and go "Settings" > "Management area settings" > "default" page. If the host name of the server that the job manager package is installed is displayed in the "Job Manager Status" section, communication should be ok.

Update

You can update the job manager package by the same procedure as the installation.

1.2.5 Send-Event package

Installation /package updates including event transmission script will be explained here.

Send-Event packages are compatible with Linux and Windows.

Extract the Kompira package and run install.sh.

Exract Kompira packages and run install.sh

Since the Send-Event script sends data to Kompira itself, you need to specify the host name or IP address of the server on which Kompira packages are installed as an argument to install.sh.

Replace <version> with the version number of Kompira. <kompira_ip> is the host name or IP address of the Kompira server.

```
$ tar zxf kompira-<version>-bin.tar.gz
$ cd kompira-<version>-bin
# ./install.sh --sendevt <kompira_ip>
[2020-09-18 01:00:27] ****:␣
↪************************************************************
[2020-09-18 01:00:27] ****: Kompira-1.6.0:
[2020-09-18 01:00:27] ****: Start: Install the Kompira
```

(continues on next page)

(continued from previous page)

```
[2020-09-18 01:00:27] ****:
[2020-09-18 01:00:27] INFO:     SYSTEM                    = CENT
[2020-09-18 01:00:27] INFO:     SYSTEM_NAME               = cent8
[2020-09-18 01:00:27] INFO:     SYSTEM_RELEASE            = CentOS Linux release 8.2.
↪2004 (Core)
[2020-09-18 01:00:27] INFO:     SYSTEM_RELEASEVER         = 8.2.2004
[2020-09-18 01:00:27] INFO:     PLATFORM_PYTHON           = /usr/libexec/platform-
↪python

...

[2020-09-18 01:00:39] ****: ------------------------------------------------------------
↪------
[2020-09-18 01:00:39] ****: Setup kompira common files.
[2020-09-18 01:00:39] ****:
[2020-09-18 01:00:39] INFO: Create log directory: /var/log/kompira
[2020-09-18 01:00:39] VERBOSE: run: install -g kompira -m 775 -d /var/log/kompira
[2020-09-18 01:00:39] VERBOSE: run: find /var/log/kompira -type f -user root -exec␣
↪chown kompira:kompira {} ;
[2020-09-18 01:00:39] INFO: Create Kompira/Kompira-sendevt configuration file.

...

[2020-09-18 01:00:39] VERBOSE: run: install -S .old -b /home/hattori/kompira-1.6.0rc1-
↪bin/.tmp.install-20200918-0100.w27C/kompira.conf -m 644 /opt/kompira/kompira.conf
[2020-09-18 01:00:39] ****:
[2020-09-18 01:00:39] ****: Finish: Install the Kompira (status=0)
[2020-09-18 01:00:39] ****:␣
↪************************************************************
```

If “Finish: Install the Kompira (status=0)” is displayed, the installation has been a success.

When the installation is completed, the kompira_sendevt will be placed under /opt/kompira/bin

```
$ /opt/kompira/bin/kompira_sendevt --version
kompira_sendevt (Kompira version 1.6.0)
```

Installation on Windows

1. **Installing Python**

   Install Python 3.6 for Windows.

   - https://www.python.org/downloads/

   Download the latest Python 3.6 installer for Windows from the above mentioned official site and install it on your Windows Operating System.

   When the installation is completed, add the environment variable path so that Python can be called from the command line.

Path	Description
`C:\Python36`	Folder that Python commands are stored
`C:\Python36\Scripts`	Folder that pip and other command types are stored

2. **Creating a Python virtual environment for Kompira**

Create an independent Python virtual environment (virtuelenv) for Kompira in `C:\Kompira`.

```
C:\> pip install virtualenv
C:\> python -m virtualenv C:\Kompira
```

3. **Create the directory for the log files**

   Create the directory `C:\var\log\kompira` as the log file output destination.

   ```
   C:\> mkdir C:\var\log\kompira
   ```

4. **Installation of the kompira_sendevt package**

   After downloading and unpacking the Kompira package on Windows, Install the Kompira_sendevt-<version>-py3-none-any.whl package with plp.exe.

   ```
   C:\> C:\Kompira\Scripts\pip.exe install Kompira_sendevt-1.6.0-py3-none-
   ↪any.whl
   Processing c:\users\kompira\documents\kompira-package\kompira_sendevt-1.6.
   ↪0-py3-none-any.whl
   Collecting amqp~=2.5.2 (from Kompira-sendevt==1.6.0)

   ...

   Installing collected packages: subprocess32, amqp, decorator, simplejson,␣
   ↪Kompira-sendevt
   Successfully installed Kompira-sendevt-1.6.0 amqp-2.5.2 decorator-4.4.2
   ```

   The Send-Event package installation is now complete. The kompira_sendevt will be placed under `C:\Kompira\Scripts`. Try running the kompira_sendevt command as follows.

   ```
   C:\> C:\Kompira\Scripts\kompira_sendevt.exe --version
   kompira_sendevt (Kompira version 1.6.0)
   ```

   If it is correctly installed, it will display the version number.

   If you add `C:\Kompira\Scripts` to the environment variable PATH, you can omit the path and execute it.

Update

With the same procedure as the installation you can update the Send-Event package.

1.3 Kompira process management

The Kompira system has multiple processes working together. Kompira's process structure will be explained below.

1.3.1 Structure of Kompira processes

The Kompira system structure processes are as follows:

Kompira daemon (kompirad) Kompira Daemon process for executing and managing job flow.

Kompira Job Manager is requested to execute remote command and receives the result.

Kompira Job Manager (kompira_jobmngrd) This is a daemon process for executing the remote command requested from the Kompira daemon.

Kompira Job Manager will connect to the remote host with ssh or winrs and execute the command when receiving the remote command from the Kompira daemon. The command execution result will be sent to the Kompira daemon.

Other processes required for the Kompira system are Apache (httpd), PostgreSQL (postgresql), RabbitMQ (rabbitmq-server), lsyncd.

Each of these processes is set up by install.sh to start automatically at machine startup.

1.3.2 Starting / stopping the Kompira daemon and Checking the state

Please start and stop the Kompira daemon as root. The user running daemon will change to Kompira automatically after startup.

For RHEL / CentOS 7x / 8x

Start the Kompira daemon on RHEL / CentOS 7x / 8x by the following command.

```
# systemctl start kompirad
```

To abort, execute the following command.

```
# systemctl stop kompirad
```

With the systemctl status command you can check the status of the Kompira daemon.

```
$ systemctl status kompirad.service
* kompirad.service - Kompira-daemon
   Loaded: loaded (/usr/lib/systemd/system/kompirad.service; enabled; vendor preset:␣
↪disabled)
   Active: active (running) since Thu 2018-07-05 16:33:02 JST; 2h 20min ago
  Process: 5277 ExecStartPre=/opt/kompira/bin/prestart_kompirad.sh (code=exited,␣
↪status=0/SUCCESS)
 Main PID: 5368 (kompirad)
   CGroup: /system.slice/kompirad.service
           `-5368 /opt/kompira/bin/python3.6 /opt/kompira/bin/kompirad

 Jul 05 16:32:32 kompira-install-test3.dev.fixpoint.co.jp systemd[1]: Starting␣
↪Kompira-daemon...
 Jul 05 16:33:02 kompira-install-test3.dev.fixpoint.co.jp systemd[1]: Started Kompira-
↪daemon.
```

When it is started, the Active: section is as **active (running)**, and when it is aborted it displays as **inactive (dead)**.

1.3.3 Starting / stopping the Kompira daemon and Checking the status.

Please start and stop the Kompira daemon as root. User to run daemon will be change to kompira are automatically after startup.

For RHEL / CentOS 7x / 8x

Start the Kompira job manager on RHEL / CentOS 7x / 8x by the following command.

```
# systemctl start kompira_jobmngrd.service
```

To abort, execute the following command.

```
# systemctl stop kompira_jobmngrd.service
```

With the status command you can check the status of the Kompira job manager.

```
$ systemctl status kompira_jobmngrd.service
* kompira_jobmngrd.service - Kompira-jobmanager
   Loaded: loaded (/usr/lib/systemd/system/kompira_jobmngrd.service; enabled; vendor␣
↪preset: disabled)
   Active: active (running) since Thu 2018-07-05 16:32:22 JST; 2h 25min ago
 Main PID: 5164 (kompira_jobmngr)
   CGroup: /system.slice/kompira_jobmngrd.service
           |-5164 /opt/kompira/bin/python3.6 /opt/kompira/bin/kompira_jobmngrd
           `-5197 /opt/kompira/bin/python3.6 /opt/kompira/bin/kompira_jobmngrd

 Jul 05 16:32:22 kompira-install-test3.dev.fixpoint.co.jp systemd[1]: Started Kompira-
↪jobmanager.
 Jul 05 16:32:22 kompira-install-test3.dev.fixpoint.co.jp systemd[1]: Starting␣
↪Kompira-jobmanager...
```

When it is started, the Active: section is as **active (running)**, and when it is aborted it displays as **inactive (dead)**.

1.3.4 Port List used by Kompira

On the server on which the Kompira package is installed, the following ports need to be open to access from outside.

Port number	Description
80/TCP	HTTP (it is unnecessary when accessing only HTTPS)
443/TCP	HTTPS (it is unnecessary when accessing only HTTP)
5672/TCP	AMQP (only when using the Job Manager package and Sned-Event package)
5405/UDP	corosync(Only with HA)
873/TCP	rsync (Only with HA)
5432/TCP,UDP	PostgreSQL (Only with HA)

1.4 Node setting

'ssh' and 'winrs' are supported to control target nodes that execute remote jobs from Kompira. To connect to UNIX/Linux machine(s), you will need to configure 'ssh node settings', and to connect to a Windows device you will need to configure 'winrs node settings'.

1.4.1 SSH node setting

When executing commands from Kompira with ssh, you should log in with ssh version 2. As for recent Linux, ssh login is ready by default so it is not necessary to configure. For details on how to enable ssh login on other nodes, refer to the manual of each operation system.

Note: Supported version of SSH is only v2. SSH v1 is not supported.

1.4.2 Winrs node setting

When executing commands from Kompira to the winrs node, WinRM setting is required on the winrs node. The supported version of WinRM is 1.1, 2.0, 3.0.

1. **Enabling remote management of WinRM**

   In order to enable WinRM, please run Windows Command Prompt as Administrator and execute `winrm quickconfig` (or `winrm qc`). When you are prompted to select y/n, i.e. "Do you want to change [y / n]?" Please enter y. Note that this operation is not needed from the second time.

   The following is an example of Windows 7, but the details of the contents displayed can be different depending on the version of Windows and the setting.

```
C:\>winrm quickconfig
...
Make these changes [y/n]? y
...
```

2. **Changing WinRM connection settings**

   To allow unencrypted communications with WinRM, run the following from a command prompt (run as administrator) as well.

```
C:\> winrm set winrm/config/service @{AllowUnencrypted="true"}
```

Note: Since Kompira Ver.1.4.10 and later, NTLM authentication is supported by default, so BASIC authentication is no longer needed.

---

3. Test job flow

   Please check the command flow to the winrs node by creating and executing the following job flow with Kompira. Please create the following job flow and run with Kompira, then check if you can command to winrs node.

```
[__host__ = '<IP address of Windows server>',
 __user__ = '<Username of Windows account>',
 __password__ = '<Password of Windows account>',
 __conntype__ = 'winrs']
-> ['ver']
-> print($RESULT)
```

It is successful if the Windows version number is displayed on the console of the job flow process.

In WinRM 2.0 and later, TCP port 5985 is used by default, but in WinRM 1.1 such as Windows Server 2008, the port number used is 80. In that case, add the port number setting `__port__ = 80`.

If you cannot connect properly, make sure that the firewall allows TCP port 5985 (or 80) to pass through, and check whether the login account settings are correct or not.

1.5 Kompira settings and log files

The following is an explanation of the directory and configuration files on the server that are standard to Kompira.

1.5.1 Kompira standard directories.

The following is a list of directories and configuration files on the server that Kompira uses as standard.

Path		Description
/opt/kompira/		
	bin/	Directory for Kompira executable file
	kompira.conf	Kompira configuration files
/var/log/kompira/		Directory for engine and job manager log files
/var/opt/kompira/		Directory of Kompira variable files
	kompira.lic	Kompira license
	html/	Online help's HTML file group
	repository/	Working directory for repository link
	upload/	Directory for attachment files
/etc/httpd/conf.d/	kompira.conf	Apache setting files

1.5.2 Kompira logs

The log files of Kompira are created under the following directory as standard.

- /var/log/kompira/

The log file name and contents generated are as follows.

Log file name	Contents
kompira.log	Request related log output
kompirad.log	Kompira daemon log output
process.log	Log output of Kompira job flow process
kompira_jobmngrd.log	Kompira Job Manager log output
kompira_sendevt.log	Log output of Send-Event command

The log files are automatically rotated, and the old log files are saved with the date added to the file name.

For the above log files, kompira_jobmngrd.log and kompira_sendevt.log the settings of rotation, such as interval and number of generations can be changed in /opt/kompira/kompira.conf.

1.5.3 Kompira configuration files

The setting items in /opt/kompira/kompira.conf are as follows.

Section name	Item name	Default value	Contents
kompira	site_id	1	Not used in this version
logging	Log output related settings		
	loglevel	INFO	Setting the log level (DEBUG, INFO, WARNING, ERROR, CRITICAL)
	logdir	/var/log/kompira	Directory of log files
	logbackup	kompirad: 7 kompira_jobmngrd: 7 kompira_sendevt: 10	Number of generations of log backup
	logmaxsz	kompirad: 0 kompira_jobmngrd: 0 kompira_sendevt: 1024*1024*1024	Maximum log file size (in bytes) Set to 0 to rotate daily
amqp-connection	RabbitMQ connection information related settings		
	server	localhost	Connection host name
	port	5672	Connection port number
	user	guest	Connection user name
	password	guest	Connection password
	ssl	false	Enabling to connect with SSL or not
	heartbeat_interval	10	Heartbeat interval (in seconds)
	max_retry	3	Maximum number of attempts to reconnect at disconnection
	retry_interval	30	Interval (in seconds) to reconnect at disconnection
agent	Settings related to job manager operation		
	name	default	Name of job manager
	script_dir	/var/tmp/kompira_jobmngrd	Temporary directory for script jobs
	remote_script_dir	~/.kompira_jobmngrd	Remote temporary directory for script jobs
	ssh_port	Not set	SSH default port
	ssh_keyfile	Not set	The default path of ssh key file
	pool_size	8	Number of concurrent process workers
	disable_cache	false	Disable SSH Connection Cache
	cache_duration	300	SSH connection cache expiration date (in seconds)
event	Settings of Send-Event		
	channel	/system/channels/Alert	Path on Kompira for event transmission channel

1.6 Data backup of Kompira

How to back up and restore data stored on Kompira.

The definitions of job flow and device information created on Kompira will be stored in the database. These data sets can be exported and imported as a file in json format.

1.6.1 Export of Kompira objects

By executing the export_data command with the following format, specified data of Kompira file system will be dumped in json format.

```
/opt/kompira/bin/manage.py export_data [options] <path>...
```

For example, to export all data under /home/guest created by Kompira to a file, execute the following command.

```
$ /opt/kompira/bin/manage.py export_data /home/guest > guest.json
```

Alternatively, by executing the export_dir command, you can dump the data below the path of the Kompira file system specified by the argument as a file on a per-object basis.

```
/opt/kompira/bin/manage.py export_dir [options] <path>...
```

1.6.2 Import of Kompira objects

You can import data with the exported file using the import_data command. The format of the import_data command is as follows.

```
/opt/kompira/bin/manage.py import_data [options] <filename>...
```

For example, to import the file guest.json exporting the /home/guest directory, execute the following command:

```
$ /opt/kompira/bin/manage.py import_data guest.json
[2014-07-25 12:34:49,576] INFO: import data: start...
[2014-07-25 12:34:49,676] INFO: home/guest: import is skipped: "/home/guest" already␣
↪exists.
[2014-07-25 12:34:49,710] INFO: home/guest/a: import is skipped: "/home/guest/a"␣
↪already exists.
[2014-07-25 12:34:49,743] INFO: home/guest/b: import is skipped: "/home/guest/b"␣
↪already exists.
[2014-07-25 12:34:49,743] INFO: import data: finished (created=0, updated=0,␣
↪skipped=3, error=0)
```

If the imported json file contains an object of a path that already exists, the import of that object will be skipped. In the above case, all three files to be imported were skipped.

You can overwrite by using the overwrite-mode option.

```
$ /opt/kompira/bin/manage.py import_data --overwrite-mode guest.json
[2014-07-25 12:39:15,685] INFO: import data: start...
[2014-07-25 12:39:15,821] INFO: import object: imported "home/guest" to "/home/guest"␣
↪(updated)
[2014-07-25 12:39:15,904] INFO: import object: imported "home/guest/a" to "/home/
↪guest/a" (updated)
```

(continues on next page)

(continued from previous page)

```
[2014-07-25 12:39:15,971] INFO: import object: imported "home/guest/b" to "/home/
↪guest/b" (updated)
[2014-07-25 12:39:15,991] INFO: import fields: /home/guest: []
[2014-07-25 12:39:16,015] INFO: import fields: /home/guest/a: ['wikitext']
[2014-07-25 12:39:16,046] INFO: import fields: /home/guest/b: ['wikitext']
[2014-07-25 12:39:16,046] INFO: import data: finished (created=0, updated=3,␣
↪skipped=0, error=0)
```

Files dumped with the export_dir command can be imported using the import_dir command.

```
/opt/kompira/bin/manage.py import_dir [options] <dirname>...
```

1.6.3 Backup

The Kompira backup procedure.

In addition to the database, Kompira uses the path listed in *Kompira standard directories.* on the server. When backing up Kompira's data, in addition to backing up the Kompira object with the export_data command, you also need to back up files on the server.

In particular, /var/opt/kompira/upload/ is the directory where the uploaded file from the attachment field on Kompira is saved, so it is a good idea to back it up there.

This is an example of backing up the Kompira object and attached file directory.

```
$ mkdir -p /tmp/kompira_backup
$ cd /tmp/kompira_backup
$ /opt/kompira/bin/manage.py export_data / --virtual-mode > backup.json
$ cp -r /var/opt/kompira/upload ./
$ cd /tmp
$ tar zcf kompira_backup.tar.gz ./kompira_backup
```

1.6.4 export_data options

The export_data command has the following options:

Options	Description
`--directory=DIRECTORY`	Specify the directory as the starting point of the exported path. (Default is '/')
`--virtual-mode`	This also outputs data contained in the virtual file system.
`--owner-mode`	This also outputs the exported user object owned by that user and the group object belonging to that user.
`-h, --help`	Print help message.

1.6.5 export_dir options

The export_dir command has the following options:

Options	Description
`--directory=DIRECTORY`	Specify the directory as the starting point of the exported path. (Default is '/')
`--property-mode`	Attributes such as 'display_name' are also output.
`--datetime-mode`	The 'created' and 'updated' are also output.
`--current=CURRENT_DIR`	Specify the output directory.
`-h, --help`	Print help message.

1.6.6 import_data options

The import_data command has the following options:

Options	Description
`--user=USER`	Set the owner of the data to be imported to USER (specify user ID).
`--directory=ORIGIN-DIR`	Specify the directory as the starting point of the import destination. (Default is '/')
`--overwrite-mode`	Overwrite existing files.
`--owner-mode`	Set the owner of the data to be imported to the export owner.
`--now-updated-mode`	Sets the current time as the 'updated' of the object.
`-h, --help`	Print help message.

1.6.7 import_dir options

The import_dir command has the following options:

Options	Description
`--user=USER`	Set the owner of the data to be imported to USER (specify user ID).
`--directory=ORIGIN-DIR`	Specify the directory as the starting point of the import destination. (Default is '/')
`--overwrite-mode`	Overwrite existing files.
`--owner-mode`	Set the owner of the data to be imported to the export owner.
`--now-updated-mode`	Sets the current time as the 'updated' of the object.
`-h, --help`	Print help message.

1.7 Kompira License

You can check the license status of Kompira using the license_info command. The format of the license_info command is as follows.

```
/opt/kompira/bin/manage.py license_info
```

The following is an example of when a license is registered.

```
$ /opt/kompira/bin/manage.py license_info
*** Kompira License Information ***
License ID:     KP-REGLM0-0000000001
Edition:        REGL
Hardware ID:    NODE:000C29FB949E
Expire date:    2015-12-31
The number of registered nodes: 0 / 100
The number of registered jobflows:      2 / 100
The number of registered scripts:       0 / 100
Licensee:       fixpoint,inc.
Signature:      dwyWvG9eKbnGxcpWfVr1H0wSybLkGL7UqB2E6d5f0jYapfTx/
↪AABJ66W3sRpK0byk+9Y724NuEZ9Rh90ySU8f2GRsIyujuVrgPloajokbdZrPFIqOlyvLkak8MAWcGJxiioPHPNd2Tv2BNOsq6bs
```

If the license is not registered, the temporary license information will be printed.

```
$ /opt/kompira/bin/manage.py license_info
*** Kompira License Information ***
License ID:     KP-TEMP-0000000000
Edition:        temporary
Hardware ID:    NODE:000C29FB949E
Expire date:    2015-01-22
The number of registered nodes: 0 / 100
The number of registered jobflows:      2 / 100
The number of registered scripts:       0 / 100
Licensee:
Signature:      None

Kompira is running with temporary license.
```

The license file path is /var/opt/kompira/kompira.lic.

Place the license file in the above path and use the license_info command to check if the license file is loaded correctly on Kompira.

This is an example of registering the license file kompira_KP-REGLM100-0000000001.lic on the command line to Kompira which no license has been registered.

```
$ cp kompira_KP-REGLM100-0000000001.lic /var/opt/kompira/kompira.lic
$ cd /var/opt/kompira
$ chown apache:apache kompira.lic
$ chmod 644 kompira.lic
$ /opt/kompira/bin/manage.py license_info
*** Kompira License Information ***
License ID:     KP-REGLM0-0000000001
Edition:        REGL
Hardware ID:    NODE:000C29FB949E
Expire date:    2017-12-31
The number of registered nodes: 0 / 100
The number of registered jobflows:      2 / 100
The number of registered scripts:       0 / 100
Licensee:       fixpoint,inc.
Signature:      dwyWvG9eKbnGxcpWfVr1H0wSybLkGL7UqB2E6d5f0jYapfTx/
↪AABJ66W3sRpK0byk+9Y724NuEZ9Rh90ySU8f2GRsIyujuVrgPloajokbdZrPFIqOlyvLkak8MAWcGJxiioPHPNd2Tv2BNOsq6bs
```

New in version 1.4.2: license_info command

See also:

License Management: You can also check and register licenses from the browser.

1.8 High Availability (HA) Management

Kompira can be operated on two servers with active-standby redundant configuration using Pacemaker / corosync. The following is an explanation of its installation, state check, failover, etc.

1.8.1 Introduction

Pacemaker monitors the resources (applications) necessary for Kompira and failover when an error is detected for the redundancy.

The list of resources Pacemaker monitors is as follows.

httpd, kompirad, kompira_jobmngrd This is a necessary process for Kompira and can only run on an active server.

RabbitMQ RabbitMQ is also a necessary process for Kompira. A process on the active server is a 'Master', and a process on the standby server is a 'Slave'.

IPaddr2 A resource for managing virtual IP addresses.

PostgreSQL This is a PostgreSQL database process. A process on the active server is Master, and a process on the standby server is Slave. PostgreSQL replication is configured so the data on the primary server and the data on the secondary server are synchronized.

lsyncd This process is for file mirroring. Attached files on the server file system will be mirrored. It runs only on active server

1.8.2 Installation

When building a redundant configuration of Kompira, after installing Kompira on each of the two servers, set up the redundant configuration on the primary server and then the secondary server.

Two servers require two network interfaces. Depending on the OS version, the network interface name may be eth0, eth1, . . . , or ens192, ens224, ...

From the following, we call the primary server (kompira-server1), and the secondary server (kompira-server2), and each server has network interfaces eth0, snd eth1. eth0 is connected to the service provisioning network and eth1 is used for heartbeat so that two servers are connected by an independent network.

When the redundant configuration is established, the primary server is in the active state and the secondary server is in the standby state.

To build a redundant configuration of Kompira, use setup_cluster.sh included in the package. In the following, the procedure for installing redundant Kompira on 2 servers after OS installation is explained.

Note: setup_cluster.sh downloads various middleware from the outside like install.sh. Please run it while you have an Internet connection available.

Setting of the primary server

After installing the Kompira package, set up the primary server by running setup_cluster.sh with the `--primary` option.

When executing setup_cluster.sh, specify the following information as an argument.

- Heartbeat network device name
- The virtual IP address (VIP) assigned to the cluster and its subnet mask prefix size

For example, to specify eth1 as the heartbeat network device, 192.168.0.100 as the virtual IP address and 24 as the subnet mask prefix size, execute the following commands.

```
$ tar zxf kompira-<version>-bin.tar.gz
$ cd kompira-<version>-bin
# ./install.sh
# ./setup_cluster.sh --primary --heartbeat-device eth1 192.168.0.100/24
```

Note: setup_cluster.sh changes the host. It is set to kompira-server 1 for the primary server and kompira-server2 for the secondary server.

Setting of the secondary server

After installing the Kompira package, set up the secondary server by executing setup_cluster.sh with the `--secondary` option.

When executing setup_cluster.sh, specify the following information as an argument. It is not necessary to specify a virtual IP address.

- Heartbeat network device name

Heartbeat network device name

```
$ tar zxf kompira-<version>-bin.tar.gz
$ cd kompira-<version>-bin
# ./install.sh
# ./setup_cluster.sh --secondary --heartbeat-device eth1
```

Note: For example, to specify eth1 as the heartbeat network device, execute the following commands.

Status check

When the installation of the primary server and the secondary server are completed, please access the following URL from a Web browser and confirm that the login screen is displayed.

```
http://192.168.0.100/
```

The URL is the virtual IP address that was set when installing the primary server. This URL will be maintained even if the primary machine fails and a failover occurs.

Also, to check the status of each resource in the redundant configuration, use the crm_mon command on the primary server or the secondary server.

```
$ crm_mon -A1
Last updated: Thu Jul 10 17:26:44 2014
Last change: Thu Jul 10 17:26:33 2014 via crm_attribute on kompira-server1
Stack: cman
Current DC: kompira-server1 - partition with quorum
Version: 1.1.10-14.el6_5.3-368c726
2 Nodes configured
9 Resources configured


Online: [ kompira-server1 kompira-server2 ]

 Resource Group: webserver
     res_vip    (ocf::heartbeat:IPaddr2):       Started kompira-server1
     res_httpd  (ocf::heartbeat:apache):        Started kompira-server1
     res_kompirad       (ocf::kompira:kompirad):        Started kompira-server1
     res_kompira_jobmngrd       (ocf::kompira:kompira_jobmngrd):        Started␣
↪kompira-server1
     res_lsyncd (lsb:lsyncd):   Started kompira-server1
 Master/Slave Set: ms_pgsql [res_pgsql]
     Masters: [ kompira-server1 ]
     Slaves: [ kompira-server2 ]
 Master/Slave Set: ms_rabbitmq [res_rabbitmq]
     Masters: [ kompira-server1 ]
     Slaves: [ kompira-server2 ]

Node Attributes:
* Node kompira-server1:
    + master-res_pgsql                 : 1000
    + master-res_rabbitmq              : 10
    + res_pgsql-data-status            : LATEST
    + res_pgsql-master-baseline        : 0000000007000090
    + res_pgsql-status                 : PRI
* Node kompira-server2:
    + master-res_pgsql                 : 100
    + master-res_rabbitmq              : 5
    + res_pgsql-data-status            : STREAMING|ASYNC
    + res_pgsql-status                 : HS:async
```

Here is the points to check in the output of the crm_mon command.

- **Resource Group**

  Only resources that are running on the active machine are printed. Everything is normal if "Started <host name of active machine>" is printed.

- **Master/Slave Set**

  Resources running on both servers are printed. It is normal if Masters is the active server host name and Slaves is the standby server host name.

- **Node Attributes**

  Detailed status of the PostgreSQL process is printed. If replication has been performed correctly, it prints in the res_pgsql-data-status line as LATEST on the active server and STREAMING | ASYNC on the standby server.

License Registration

In a redundant configuration, you will need to register license files for both active and standby servers.

Please follow the procedure in *Kompira License* and register the license file for each server.

1.8.3 HA stop and start

How to stop and start Kompira operating in a HA:

First of all, use the crm_mon command to see which of the two servers is acting as active. In the resource part of the crm_mon command result the active server shows, “Started” and “Masters”.

The following explanation assumes that kompira-server1 is in the active state.

In principle, to stop servers, stop the standby server first and then stop the active server, to start servers, start the active server first and then start the standby server second.

This is because if the active server is stopped first, the standby server judges that an error has occurred in the active server and a failover process will be performed. If it fails over by mistake , refer to *Failover and fail back behavior*.

Stop HA configurtion

First, stop the Pacemaker process on the secondary server (kompira-server2).

```
# pcs cluster stop
Stopping Cluster (pacemaker)...
Stopping Cluster (cman)...
```

After confirming that the service has stopped, do the same thing on the primary server (kompira-server1). The `--force` option is required to stop the last one of the HA configuration.

```
# pcs cluster stop --force
Stopping Cluster (pacemaker)...
Stopping Cluster (cman)...
```

This stops monitoring resources by Pacemaker/corosync. Please note that the crm_mon command can not be executed when the pacemaker process is stopped.

To not only stop the process but to also shut down the server OS, the above process is not needed. However, please shut down the standby server first and then the active server.

Start HA configuration

To start up, follow the procedure opposite to stop. First, start the Pacemaker process on the primary server (kompira-server1).

```
# pcs cluster start
Starting Cluster...
```

When the pacemaker process started up, the resources registered in pacemaker will start sequentially. Execute the crm_mon command and wait until all resources are started.

When the resources were started, start the Pacemaker process on the secondary server (kompira-server2).

```
# pcs cluster start
pcs cluster start
```

This completes start up of the HA configuration.

When you boot server OS not only starting processes, the above processing is not needed. The Pacemaker service is set to auto start up.

Please start the active server first and after confirming startup is completed, then start the standby server.

1.8.4 Failover and fail back behavior

If any failure occurs on the active server, the failover will be automatically performed and the standby server will be promoted to the active status.

Below is the crm_mon command result on kompira-server2 after shutting down of kompira-server1 which was in the active state.

```
$ crm_mon -A1
Last updated: Thu Jul 10 17:40:36 2014
Last change: Thu Jul 10 17:36:24 2014 via crm_attribute on kompira-server2
Stack: cman
Current DC: kompira-server2 - partition WITHOUT quorum
Version: 1.1.10-14.el6_5.3-368c726
2 Nodes configured
9 Resources configured


Online: [ kompira-server2 ]
OFFLINE: [ kompira-server1 ]

 Resource Group: webserver
     res_vip    (ocf::heartbeat:IPaddr2):       Started kompira-server2
     res_httpd  (ocf::heartbeat:apache):        Started kompira-server2
     res_kompirad       (ocf::kompira:kompirad):        Started kompira-server2
     res_kompira_jobmngrd       (ocf::kompira:kompira_jobmngrd):        Started␣
↪kompira-server2
     res_lsyncd (lsb:lsyncd):   Started kompira-server2
 Master/Slave Set: ms_pgsql [res_pgsql]
     Masters: [ kompira-server2 ]
     Stopped: [ kompira-server1 ]
 Master/Slave Set: ms_rabbitmq [res_rabbitmq]
     Masters: [ kompira-server2 ]
     Stopped: [ kompira-server1 ]

Node Attributes:
* Node kompira-server2:
    + master-res_pgsql                  : 1000
    + master-res_rabbitmq               : 10
    + res_pgsql-data-status             : LATEST
    + res_pgsql-master-baseline         : 000000000A000090
    + res_pgsql-status                  : PRI
```

You can see kompira-server1 is OFFLINE, and each of the resources are running on kompira-server2.

In the following, the procedures when kompira-server1 is recoverable, and unrecoverable separately will be explained.

When the server is recoverable

Here is the procedure when the kompira-server1 can be started normally.

When you have started kompira-server1, the status will be as follows

```
Last updated: Thu Jul 10 18:02:02 2014
Last change: Thu Jul 10 17:36:24 2014 via crm_attribute on kompira-server2
Stack: cman
Current DC: kompira-server2 - partition with quorum
Version: 1.1.10-14.el6_5.3-368c726
2 Nodes configured
9 Resources configured


Online: [ kompira-server1 kompira-server2 ]

 Resource Group: webserver
     res_vip    (ocf::heartbeat:IPaddr2):       Started kompira-server2
     res_httpd  (ocf::heartbeat:apache):        Started kompira-server2
     res_kompirad       (ocf::kompira:kompirad):    Started kompira-server2
     res_kompira_jobmngrd       (ocf::kompira:kompira_jobmngrd):    Started kompira-
↪server2
     res_lsyncd (lsb:lsyncd):   Started kompira-server2
 Master/Slave Set: ms_pgsql [res_pgsql]
     Masters: [ kompira-server2 ]
     Stopped: [ kompira-server1 ]
 Master/Slave Set: ms_rabbitmq [res_rabbitmq]
     Masters: [ kompira-server2 ]
     Slaves: [ kompira-server1 ]

Node Attributes:
* Node kompira-server1:
    + master-res_pgsql                  : -INFINITY
    + master-res_rabbitmq               : 5
    + res_pgsql-data-status             : DISCONNECT
    + res_pgsql-status                  : STOP
* Node kompira-server2:
    + master-res_pgsql                  : 1000
    + master-res_rabbitmq               : 10
    + res_pgsql-data-status             : LATEST
    + res_pgsql-master-baseline         : 000000000A000090
    + res_pgsql-status                  : PRI

Failed actions:
res_pgsql_start_0 on kompira-server1 'unknown error' (1): call=40, status=complete,␣
↪last-rc-change='Thu Jul 10 17:59:25 2014', queued=776ms, exec=0ms
```

In kompira-server1, the database is not running and it is not in a normal state as a standby server.

In order to complete the setup as a standby server, use sync_master.sh in the kompira package on the standby server. sync_master.sh will copy the database of the active server to the standby server, sets up replication, and starts the database process.

```
# /opt/kompira/bin/sync_master.sh
[2018-07-05 22:47:09] ****:␣
↪************************************************************
[2018-07-05 22:47:09] ****: Kompira-1.5.0:
```

(continues on next page)

(continued from previous page)

```
[2018-07-05 22:47:09] ****: Start: Sync with the Master
...
[2018-07-05 22:47:15] INFO: Waiting for the resources to stabilize.
    RabbitMQ  | PostgreSQL                          | description
    #NA       | /DISCONNECT/#NA                     |
    #NA       | /DISCONNECT/#NA                     |
...
[2018-07-05 22:47:32] INFO: PostgreSQL on this node needs to be synchronized with the␣
↪Master.
[2018-07-05 22:47:32] INFO: RabbitMQ on this node is running as a Slave.
[2018-07-05 22:47:32] INFO: Enter the pacemaker in maintenance mode.
[2018-07-05 22:47:32] VERBOSE: run: pcs property set maintenance-mode=true
[2018-07-05 22:47:33] VERBOSE: run: rm -f /var/lib/pgsql/9.6/tmp/PGSQL.lock
[2018-07-05 22:47:33] VERBOSE: run: rm -rf /var/lib/pgsql/9.6/data.old
[2018-07-05 22:47:33] VERBOSE: run: mv -f /var/lib/pgsql/9.6/data /var/lib/pgsql/9.6/
↪data.old
[2018-07-05 22:47:33] VERBOSE: run: sudo -u postgres pg_basebackup -h ha-kompira-
↪other -U repl -D /var/lib/pgsql/9.6/data -X stream -P
33488/33488 kB (100%), 1/1 tablespace
[2018-07-05 22:47:35] INFO: Reset the failcount of RabbitMQ.
[2018-07-05 22:47:35] VERBOSE: run: crm_failcount -r res_rabbitmq -D
...
[2018-07-05 22:47:36] INFO: Waiting for the resources to stabilize.
    RabbitMQ  | PostgreSQL                          | description
    5         | STOP/DISCONNECT/-INFINITY           | RabbitMQ as demoted as a Slave.
    5         | STOP/DISCONNECT/-INFINITY           |
    5         | STOP/DISCONNECT/-INFINITY           |
    5         | STOP/DISCONNECT/-INFINITY           |
    5         | STOP/DISCONNECT/-INFINITY           |
    5         | HS:alone/DISCONNECT/-INFINITY       |
    5         | HS:alone/STREAMING|ASYNC/-INFINITY  |
    5         | HS:alone/STREAMING|ASYNC/100        |
    5         | HS:async/STREAMING|ASYNC/100        | PostgreSQL has demoted as a␣
↪Slave.
[2018-07-05 22:47:45] INFO: Resources stablized.
...
[2018-07-05 22:47:48] ****:
[2018-07-05 22:47:48] ****: Finish: Sync with the Master (status=0)
[2018-07-05 22:47:48] ****:␣
↪************************************************************
```

Note: sync_master.sh saves all data related to the database to /var/lib/pgsql/9.6/data.old and gets the data of the active server. (Including log files)

After executing sync_master.sh, if you call the crm_mon command, you can confirm that state of res_pgsql-data-status of kompira-server1 has become STREAMING | ASYNC.

```
$ crm_mon -A1
Last updated: Thu Jul 10 19:03:25 2014
Last change: Thu Jul 10 19:03:16 2014 via crm_attribute on kompira-server2
Stack: cman
Current DC: kompira-server2 - partition with quorum
Version: 1.1.10-14.el6_5.3-368c726
2 Nodes configured
9 Resources configured
```

(continues on next page)

(continued from previous page)

```
Online: [ kompira-server1 kompira-server2 ]

 Resource Group: webserver
     res_vip    (ocf::heartbeat:IPaddr2):   Started kompira-server2
     res_httpd  (ocf::heartbeat:apache):    Started kompira-server2
     res_kompirad       (ocf::kompira:kompirad):    Started kompira-server2
     res_kompira_jobmngrd       (ocf::kompira:kompira_jobmngrd):    Started kompira-
↪server2
     res_lsyncd (lsb:lsyncd):   Started kompira-server2
 Master/Slave Set: ms_pgsql [res_pgsql]
     Masters: [ kompira-server2 ]
     Slaves: [ kompira-server1 ]
 Master/Slave Set: ms_rabbitmq [res_rabbitmq]
     Masters: [ kompira-server2 ]
     Slaves: [ kompira-server1 ]

Node Attributes:
* Node kompira-server1:
    + master-res_pgsql                  : 100
    + master-res_rabbitmq               : 5
    + res_pgsql-data-status             : STREAMING|ASYNC
    + res_pgsql-status                  : HS:async
* Node kompira-server2:
    + master-res_pgsql                  : 1000
    + master-res_rabbitmq               : 10
    + res_pgsql-data-status             : LATEST
    + res_pgsql-master-baseline         : 0000000004000090
    + res_pgsql-status                  : PRI
```

When the server is unrecoverable

This is the procedure when you need to shutdown the server and replace failing hardwarelure, prepare the server with OS installed and set it as a standby state.

In a HA configuration, it is necessary to register a license file on each of the active server and the standby server. Execute install.sh, then use the license_info command to check the hardware ID and register the license file.

```
$ tar zxf kompira-<version>-bin.tar.gz
$ cd kompira-<version>-bin
# ./install.sh

$ cp kompira_KP-EVALM100-0000000001.lic /var/opt/kompira/kompira.lic
$ cd /var/opt/kompira
$ chown apache:apache kompira.lic
$ /opt/kompira/bin/manage.py license_info

# ./setup_cluster.sh --primary --slave-mode
```

The above command is an example of setting up kompira-server1 in a standby state.

In a HA configuration, it is necessary to register a license file on each of the active servers and standby servers, so you will need to register the license file before running setup_cluster.sh.

When you run setup_cluster.sh to add kompira-server2 instead of kompira-server1, use the `--secondary` option instead of the `--primary` option.

When the process of setup_cluster.sh is completed, please refer to *Status check* to know how to check the status.

See also:

Kompira License

1.8.5 setup_cluster.sh Options

Below is a list of instructions on how to setup_cluster.sh.

Options	Default value	Description
`--primary`	true (specified)	Start setup as a primary
`--secondary`	false (not specified)	Start setup as secondary
`--heartbeat-device=DEVICE`		Specify the network device for heartbeat.
`--master-mode`		Setup as an active state.
`--slave-mode`		Set up as a standby state.
`--without-vip`		Setup without VIP configuration. (You will need to setup LB with ACT/SBY monitoring separately.)
`--without-jobmanager`		Setup without job manager configuration.
`--hostname-prefix=PREFIX_NAME`	kompira-server	Specify the host name prefix.
`--skip-hostname-setup`		The host name is not set. (It is necessary to set the host name in advance to resolve the host name)
`--heartbeat-netaddr=NETWORK_ADDRESS`	192.168.99.0	Specify the network address to be set for the heartbeat interface.
`--manual-heartbeat`		Manually configure the network for heartbeat. You will need to specify `--heartbeat-primary` and `--heartbeat-secondary` and ignore `--heartbeat-netaddr`. Heartbeat runs in unicast mode.
`--heartbeat-primary=NETWORK_ADDRESS`		Specify the primary IP address.
`--heartbeat-secondary=NETWORK_ADDRESS`		Specify the IP address of the secondary.
`--cluster-device=DEVICE`		Specify the network device to be assigned the VIP.
`--help`		Print help message.

CHAPTER

TWO

OPERATION GUIDE

Author Kompira development team

2.1 Introduction

In this manual, information about using Kompira's functions through the web user interface (WebUI) provided by Kompira, will be explained.

2.2 Login and logout

You can access the login screen of Kompira by accessing the following URL.

```
https://<Hostname or ipaddress of Kompira server>/
```

Please enter your user name and password to login to the Kompira login screen.

For a list of available default users, refert to: *User management*.

After logging in, you can select Home, File System, Task List, Incident List, *Process Management*, *Scheduler*, *Settings* and/or Help from the top half of the menu.

The name of the user currently logged in is displayed in the upper right corner of the screen, and you can log out from there.

Note: The login information will be saved in the browser's cookies. Login information cookies expire after 2 weeks. So after the expiry date, you will need to log in again.

2.3 Kompira file system

Kompira defined Information, such as job flow definitions and node information, are centrally managed on the Kompira file system as Kompira objects.

Below, we explain the settings and values not dependent on Kompira object type.

2.3.1 Names of object

The name of the Kompira object can be freely named within the following rules.

- You can use alphabetical and numerical characters, as well as underscores ("_"), and Japanese characters.
- The first character must be a number
- It is case-sensitive
- Object name length must be within 128 characters
- Absolute path length must be within 1024 characters

2.3.2 Object Properties

All Kompira objects have properties, and the object owner or root users can edit each item of the property.

Here is the list of items that can be set in the properties.

Field	Description
Display name	Name used to display the object. (It is different from the object name)
Description	A description of the object.
Owner	Owner of the object
User permissions	Access permission list given to users.
Group permission	The permission list given to the group.

In the section on user permissions and group permissions, you can set access permissions for each user and group.

If you want to set common access permissions for all users, it is a good idea to use the other group to which all users belong.

Note: Properties can be edited only by the owner of the object or root users, regardless of the permission setting. A user with writing permission can edit the contents of an object, but be aware that properties cannot be edited.

2.3.3 Object permissions

All Kompira objects have permission settings.

Here is a list of access permission types that the Kompira objects have.

Permission type	Description
Read	Grants the capability to read the contents of the object. An attempt to move to the path of an unauthorized object will result in an error.
Write	Grants the capability to edit the contents of the object. If you do not have writing permission on a directory or table object, you can not add new objects.
Execute	Grants the capability to execute objects It is a permission type valid only for executable objects (job flow and script job).

For root users, all access is allowed, even if they are not explicitly specified.

Object permission settings can be edited from *Object Properties* .

Note: When adding an object to a directory or table object, permission settings are not inherited.

2.4 Kompira object

There are various kinds of objects created on the Kompira file system, such as job flow and node information. These are specified by the type object on the Kompira file system. For predefined type objects, you can refer to the list in /system/types.

In the current version, the type objects shown below are defined as standard.

Type name	Description
TypeObject	An object for defining type objects. When you create a type object, you can create an object of the type you created.
Directory	An object that can store multiple objects.
Table	An object that can store multiple objects of the same type.
Jobflow	An object that can write and execute job flow.
ScriptJob	An object that can write and execute scripts.
NodeInfo	An object that can store information for specifying a node, such as server IP address or SSH port number.
AccountInfo	An object that can store account information for remote login.
Environment	An object that can store environment information in key-value format.
Template	An object that can store template text used in tasks.
MailTemplate	An object that can store the template text used for sending mail.
Wiki	An object that can create Creole format Wiki pages.
AttachedFile	An object that can save arbitrary files.
Form	An object that can create a user input form.
Config	An object that can create a setting form.
Channel	An object with a queue that can store messages. It can be used for sending and receiving messages.
MailChannel	A channel that can receive email from the IMAP server.
Repository	An object that defines information for linking with the version control system.
Library	An object that defines a Python library that can be called from a job flow.
Virtual	Object for defining virtual objects. Process list (/process) and task list (/task) are defined as virtual objects. This is not used in general.
Realm	An object for defining the area managed by the job manager. This will be created under the management area list object, and this will not be created under the normal directory or table.
License	An object for registering a license file. This is a special object used in the system, it will not be created anew.

Each type object defines its own field and its type. For details on what fields are defined , please refer to: *Built-in objects*

In the following we will introduce some typical Kompira objects and explain how to use them.

2.4.1 Directory

A directory is a Kompira object that can contain several different types of objects.

From the directory object, you can do the following operations.

Operation	Description
Create New	Create an object. When creating, you need to specify a type object.
Brows	Move to the page of the stored object.
Edit	Edit the contents of the object.
Change name	Change the name of the object. If the display name of the property is the same as the object name, the display name will be also be changed at the same time.
Move	Move the object.
Copy	Copy the object to another directory.
Delete	Delete the object.
Export	Export the selected object. When an object is not selected, all objects under the directory will be exported.
Import	Import the object into the selected directory from the file. When the directory is not selected, the object will be imported under the current directory.
Property	Edit the properties of the object.

When creating and editing, it will move to the editing screen corresponding to Kompira objects.

Multiple objects can be Moved, Copied or Deleted at one time.

Importing and Exporting objects can only be done by the owner of the directory or root users.

2.4.2 Table

A table is a Kompira object that can store multiple objects like a directory. However, it differs from the directory in that only one type of object can be stored.

When creating a table object, first select the type object and the fields in that type object. In the created table object, in addition to the information displayed in the directory object, the field information selected at the time of creation will be displayed.

By using the table object you can view all bundled fields of stored objects.

2.4.3 Job flow

The job flow can be described and executed from the job flow object. For details on syntax of the job flow, refer to: *Kompira Tutorial* . For details on the job flow language, refer to: *Kompira Jobflow Language Reference*

Job flow execution

When you write and save the job flow, the execution button of the job flow becomes effective. When you press the execute button, the job flow starts and the process details screen will be displayed.

Note: If there is a syntax error in the job flow or the Kompira engine is stopped, the execution button of the job flow will be invalid.

The following options can be selected to run a job flow.

Option name	Contents
Step mode	This mode is used when debugging a job flow. Before the command is executed, the job flow is paused and the contents of the execution command can be confirmed.
Checkpoint mode	Checkpoint mode is a mode for saving the execution status of job flow. If the Kompira server stopped abnormally during job flow execution, the job flow process can be resumed from the saved checkpoint status.
Monitoring mode	Specify the execution monitoring mode of the job flow. When the job flow is completed or abnormally stopped, a mail is sent to the mail address of the user who executed the job flow.

2.4.4 Script job

When you create a script job, you can run scripts written in languages such as Bash, Perl, Ruby, Python on a remote server.

Script edit

Pressing the Edit button will take you to the script edit screen. Write a script to execute it in the text area of the source.

When running a script on a Unix type OS such as Linux, please write (shebang) as the top line of the script.

Example

```
#!/bin/bash
echo hello
```

When executing a script on a Windows OS, you need to specify an extension. Please specify the following extensions according to the type of script.

Script	Extension
Batch file	bat
VBScript	vbs
JScript	js
PowerShell script	ps1

How to execute a script

When you press the execute button, the script's execution will be started and the process details screen is displayed. When the script is completed, the results of the exit status, standard output, and standard error output are printed on the console.

Command parameters can be entered in the text field to the right of the execute button. Multiple command arguments can be passed by separating them with spaces.

For the execution node specify the remote server on which the script is executed. If not specified, the script is executed on the local server on which the job manager is running.

For the execution account, specify the user's credential when logging in to the remote server.

New in version 1.4.0: The function to execute script jobs directly from the browser has been added.

2.4.5 Repository

It is possible to synchronize the Kompira directory and the repository on the distributed version control system (DVCS) by creating repository objects. You can import objects from the remote repository into the specified Kompira directory, or conversely save the created Kompira objects on the remote repository. This enables version control of Kompira's job flow and script jobs. It also makes it easy to share job flows across multiple Kompira.

How to set up repository

On the repository object edit screen, set the following items and save them.

Setting items	Contents
URL	Specify the URL of the remote repository.
Repository type	Specify the type of remote repository. (Only mercurial can be selected in the current version)
Port number	Specify this when the port number of the repository server is different from the default.
User name	Specify the user name of the account accessing the remote repository.
Password	Specify the password for the account that will access the remote repository.
Directory	Specify the directory object of Kompira to be synchronized.

Note: Remote repository must be created in advance.

Initialization

When necessary repository setting items are entered, the initialization button will be activated from the repository screen. Pressing the Initialize button initializes the local repository on the Kompira server and populates the Kompira directory object with the contents of the remote repository.

Push

The push button will be effective after initialization. To send updated information of the Kompira directory to the remote repository, press the push button.

Pull

The pull button will be effective after initialization. To update information on the remote repository to Kompira, press the pull button.

2.5 Process Management

We will explain the process for managing the execution state of job flows and script jobs.

A process is created when a job flow and a script job are executed. For details on starting execution, please refer to: *Job flow* and *Script job*

2.5.1 Process list

On the process list screen, you can check the list of processes that are being executed or were executed in the past.

By default, processes with process statuses NEW (New), READY (Executable), RUNNING (Running), WAITING (Waiting for Input or Command Completion) are displayed with [Running Process] selected.

When process status is DONE (completed) or ABORTED (abnormal termination), the process has already ended. If you want to check these, please select [All processes].

For processes that have already been executed, you can delete them from the list screen.

Note: Normal users can view only the processes that they themselves executed. Root users can view all processes.

2.5.2 Process details

On the process details screen, you can check and control the execution status of processes.

The buttons and forms in the process detail screen are as follows:

Terminate

Stop execution of the process. Terminated processes will show an ABORTED (abnormal termination) status and cannot be restarted.

If there is a child process running, the status of the child process will also be ABORTED (abnormal termination).

Suspend

Temporarily suspend execution of the process.

If there is a child process running, the child process will also be paused.

Resume

Resume the suspended process.

If there is a stopped child process, the child process also restarts.

Console

Output shown during process execution.

In the case of a job flow, messages of print statement, execution result of a remote command, a stack trace at error, etc. are displayed.

For script jobs, the exit status of the script, standard output, and standard error output are displayed.

Note: The maximum console buffer size is limited to 64 KB. Please note that if there is output of 64 KB or more, only the first 64 KB message can be printed.

Job flow / Script

The job flow and script executed will be displayed.

In the case of a job flow, the line currently being executed is also displayed.

Child process list

You can check the list of child processes on the screen that is displayed only when the job flow is executed.

A child process is created when you execute a job flow that creates a child process using fork or pfor syntax.

2.6 Scheduler

By adding the job flow and script job created on Kompira in the scheduler, you can run the job periodically.

Here is the list of items that can be set with the scheduler.

Field	Default value	Description
Schedule name	Nil	Name of schedule
Description	Nil	A description about the schedule
User		User who runs the job
Job		Job run by the scheduler
Year	*	Scheduled year (4 digit number)
Month	*	Scheduled month (1-12)
Date	*	Scheduled date (1-31)
ISO week number	*	Scheduled week number (1-53) Week number defied in ISO 8601
Day or day number	*	Day of the week (0 (Monday) - 6 (Sunday), or mon, tue, wed, thu, fri, sat, sun)
Hour	*	Scheduled hour (0-23)
Minute	*	Schduled minute (0-59)
Disable schedule	false (unchecked)	If true (checked), it will not run job

2.6.1 Date and time setting field format

The date and time setting field can be used the same format as Unix cron as follows.

Format	Field	Description
*	All	Run on each value
*/a	All	Run every (a).
a-b	All	Run every (a-b)
a-b/c	All	Run every (a-b) and (c).
xth y	Day	Run at the (x)th y (day) of the month.
last x	Day	Run on the last (x) day of the month.
last	Day	Run on the last day of the month.
x,y,z	All	Run with condition x, y or z (any combination of the above formats can be used)

Note: In the above format, be careful not to put a space next to ‘,’, ‘/’, ‘-‘.

Example 1: Run at 0:00 on the first Monday and last Friday of December every year:

```
Month: 12
Day: 1st mon,last fri
```

Example 2: Run at 12:30 of 15th-20th in April and August 2012:

```
Year: 2012
Month: 4,8
Day: 15-20
Hour: 12
Minute: 30
```

Example 3: Run every hour on weekdays:

```
Day of week: mon-fri
Hour: *
```

Example 4: Run at 0:00 on January 1st every year:

```
Year: *
```

2.7 Settings

Below is an explanation of various settings that you can set from the "Settings" tab at the top of the Kompira screen.

2.7.1 User management

You can check the list of users registered on Kompira.

Here is the list of initial users.

User name	Pass-word	Description
guest	guest	Guest user
root	root	root user
admin	admin	Administrative user. All objects are accessible regardless of access permission settings. By default it is a disabled user.

When creating a new user, /home/<username> directory is created automatically as the home directory.

General users can only edit their own user information. Only root users can edit all users information.

Here is the list of items that can be set for each user.

Field	Description
User name	Name used to identify users in the system
Surname	User's surname
Name	User's first name
Email	User's email address
Group	Group user belongs to.
Active	If false (unchecked), it will not allow user login
Home	The page that displays first when the user logs in
Environment variable	Environment variable object automatically loaded when running a job flow

Note: Guest, root, and admin users can not be deleted.

2.7.2 Group management

You can check the list of groups registered on Kompira.

Here is the initial group list.

Group name	Description
other	All users on Kompira belong to other
wheel	Users with root privilege belong to wheel.

Group information can only be edited by root users.

Note:

- You can not delete other and wheel groups.
- As mentioned above, to be able to see what group other users belong to can be done regardless of their settings. That means that user settings belonging to the other group are ignored.

2.7.3 Management area setting

A management area is a network area managed by each job manager.

When using multiple job managers in Kompira, Each Job Manager can specify the area, such as the job manager A will access 192.168.1.x and the Job Manager B will access 192.168.2.x.

Here is the list of items that can be set for each management area.

Field	Description
Display name	Specify the display name of the management area
Description	Describe the management area
Disable	Set this to temporarily disable the target management area
Range	Specify the range of the management area by IP address or host name. You can specify more than one, and wildcard (*) can also be used.

Only root users can edit the management area information.

By default, there is a management area named default that has it's range set to '*'. In this case, all remote commands are executed by the job manager of the default management area.

If you use Kompira with only one job manager, or if you do not need to set a management area for each job manager, you do not need to change the management area setting.

Job manager status check

In the management area setting screen, you can check the operation status of the job manager registered in each management area.

The following items are displayed as job manager status.

Value	Description
Host name	Name of the host on which the job manager is running
Process ID	The process ID of the job manager process (kompira_jobmngrd)
Version	Job Manager's Kompira Version
Status	Job manager's operating status ('Active' or 'Down')

When the status is [active], the job manager can communicate with Kompira, and remote command can be executed.

2.7.4 System Settings

On the system setting screen, you can configure the entire Kompira system.

Here is the list of setting items.

Item name (Key name)	Description
Server URL (serverUrl)	URL of Kompira server
Administrator email address (adminEmail)	Set the mail address of the administrator of the Kompira server. It is used as the default 'from' address when sending mail.
Successful email template(doneMailTemplate)	The email template used when the job flow completed normally.
Unsuccessful email template (abortMailTemplate)	The email template used when the job flow completed unsuccessfuly.
SMTP server name (SMTPServer)	Sets the SMTP server name for sending email. If omitted, the SMTP server of localhost is used.
SMTP port number (SMTPPort)	Set the port number of the SMTP server for sending mail. The default is 25.
SMTP user name (SMTPUser)	If SMTP authentication is required, set the user name.
SMTP password (SMTPPassword)	If SMTP authentication is required, set the password.
Use SMTP TLS (SMTPUseTLS)	Check this if SMTP server uses TLS.
Use SMTP SSL (SMTPUseSSL)	Check this if SMTP server uses SSL (SMTPS).

Note: Since the system setting (/system/config) is a setting type object (*Settings (Config)*) , it can refer to the data dictionary of setting's dictionary data from the job flow with key name.

2.7.5 Startup job flow

In the startup directory (/system/startup), you can set a startup job flow that starts automatically when the Kompira server starts up.

2.7.6 License Management

You can check the Kompira license

Here is the list of items that can be checked on the license management screen.

Field	Description
License ID	Unique ID of license file
Edition	License type
Hardware ID	Kompira server's Hardware unique ID
Expiration date	License expiration date
Number of registered nodes	The number of nodes that have been connected from the job flow Select Reset to delete connection history
Number of job flows	Number of job flows registered as objects
Number of scripts	Number of script jobs registered as objects
User	licensed user
signature	License file signature

The number of registered nodes, the number of job flows, and the number of scripts are displayed together with the maximum number according to the license.

If the license file is not registered, Kompira will operate using a temporary license. You can use a temporary license for up to one week after Kompira installation.

License of registration

Press the edit button on the right side of the license management screen to go to the license file upload screen.

Select the license file by pressing "Select file", and press the "Save" button to register the license.

Note: The license file is saved in /var/opt/kompira/kompira.lic. By placing the license file directly in the above path, you can register the license without accessing the license management screen.

2.8 Troubleshooting

A list of errors, causes and ways to troubleshoot them when you operate Kompira via a browser.

2.8.1 "The number of Jobflows has exceeded the limit", "The number of ScriptJobs has exceeded the limit"

The number of job flows and script job objects that can be created are controlled by the Kompira license.

If you attempt to create an object beyond the limit set by the license, an error message will be displayed and the creation of the object will fail.

Please check the available object numbers from the license management page.

2.8.2 "Kompira engine has stopped"

When the kompirad process has stopped this message is displayed.

Please check the log file under /var/log/kompira and start the kompirad process.

See also:

Starting / stopping the Kompira daemon and Checking the state, *Kompira logs*

2.8.3 Database connection error

This message is displayed when the database cannot be connected to.

You can check the status of the database process and restart it by the following commands.

RHEL/CentOS 7.x

```
# systemctl status postgresql-9.6.service
# systemctl restart postgresql-9.6.service
```

RHEL/CentOS 6.x

```
# /etc/init.d/postgresql-9.6 status
# /etc/init.d/postgresql-9.6 restart
```

Note: For Amazon Linux, the command name is postgresql96.

2.8.4 Internal error

This will be displayed when an unexpected error occurs inside Kompira.

Check the logs under /var/log/kompira and then please contact us at support@kompira.jp

2.8.5 kompira_dump.sh Information collection and support inquiries

In order to solve any problems you are having on Kompira, it may be necessary to check various information sources such as various log files and setting files.

Run `/opt/kompira/bin/kompira_dump.sh` as root on the Kompira server and this will automatically collect useful information to solve the problem. In addition, since the database dump is included, the file size can be large. Please make sure there is enough free space to run the script.

```
$ sudo /opt/kompira/bin/kompira_dump.sh
2014-11-18 15:18:52 # mkdir /home/ec2-user/kompira_dump-20141118-151852
###
### kompira_dump ver 1.0.0
### dump started: 2014-11-18 15:18:52
###
========== system ==========
2014-11-18 15:18:52 # mkdir /home/ec2-user/kompira_dump-20141118-151852/system
2014-11-18 15:18:52 # cp -a /etc/os-release /etc/system-release ./
2014-11-18 15:18:52 # printenv
2014-11-18 15:18:52 # who -aH
    :
    :
    :
========== kompira ==========
2014-11-18 15:19:09 # mkdir /home/ec2-user/kompira_dump-20141118-151852/kompira
2014-11-18 15:19:09 # /opt/kompira/bin/kompirad --version
2014-11-18 15:19:09 # /opt/kompira/bin/manage.py license_info
2014-11-18 15:19:10 # /opt/kompira/bin/manage.py dumpdata -a
2014-11-18 15:19:16 # cp -a /opt/kompira/kompira.conf ./
2014-11-18 15:19:16 # cp -a /var/opt/kompira/kompira.lic ./
2014-11-18 15:19:16 # tar -cf - /var/log/kompira
tar: Removing leading `/' from member names
---------- kompira ----------
###
### dump finished: 2014-11-18 15:19:16
###
compressing...
/home/ec2-user/kompira_dump-20141118-151852.tar.gz
```

In the last line you will see a file that summarizes the collection results (kompira_dump-20141118-151852.tar.gz in the example above). Please attach this file along with a description of the problem and email it to: support@kompira.jp

Please note that this .tar.gz file is not encrypted, so please treat it according to your security policy.

Information not collected

kompira_dump.sh does not collect confidential information such as the following.

- Account password information set in Kompira server

Information collected

kompira_dump.sh collects the following information.

- **System information**
  - Process information (ps, top, etc.)
  - Service information (service, chkconfig, etc.)
  - Installed package information (yum, rpm, pip etc)
  - Kernel information (sysctl, lsmod, /proc/{version,*info,*stat}, etc.)
  - Log files (/var/log/{dmesg,messages} etc)
- **Network information**
  - Interface information (ip link, ip addr, ip route, etc.)
  - Firewall information (iptables -L etc)
  - Network status (netstat, traceroute, etc.)
- **Information on Apache**
  - Service state (service httpd status, etc.)
  - Log files (/var/log/httpd/)
  - Configuration files (/etc/httpd)
- **Information on RabbitMQ**
  - Service status (service rabbitmq-server status)
  - Log files (/var/log/rabbitmq/)
- **Information on PostgreSQL**
  - Service status (service postgresql-<pgver> status)
  - Log files (/var/lib/pgsql/<pgver>/data/{pg_log,pgstartup.log})
- **Information on Kompira**
  - Version (kompirad –version)
  - License information (manage.py license_info, etc.)
  - Database dump (manage.py dumpdata -a)
  - Configuration file (/opt/kompira/kompira.conf)
  - Log files (/var/log/kompira/)

Note: Please note that because it contains a Kompira database dump, **node information and account / password information stored on Kompira objects and job flows are included**.

CHAPTER

THREE

KOMPIRA TUTORIAL

Author Kompira development team

3.1 Introduction

In this tutorial, the language used by Kompira to describe the job flow will be introduced.

For specifications of the Kompira standard object, refer to *Kompira Standard Library*. Alternatively, *Kompira Jobflow Language Reference* can also be used and contains more accurate definitions of the terminology.

This tutorial is not an exhaustive guide describing all the functions of Kompira. However, if you have a simple job flow then by reading this tutorial you will likely be able to understand and learn about Kompira's main functions, usage and special features.

3.2 Initiate the job flow

3.2.1 Hello World

The first job flow is simple. It is to display "Hello World" on the console.

```
print("Hello World")
```

When you run this job flow, you should see the following output in the console.

```
Hello World
```

Note: If there is a syntax error in the job flow, you will not be able to run it even when you save it. If the Run button has not been pressed, correct the error in the job flow and save it again.

In Kompira's job flow language, **job** represents a singular process in a typical execution.

In the above example, `print()` is one of the **built-in jobs** of the job flow, and outputs the character string given as an argument in the parentheses to the console. For details, see *print*.

3.2.2 How to write a comment

In the job flow, anything written from a hash tag # until the end of the line becomes a comment. A comment can be written at the beginning of a line, or even after a job. However, hash tags appearing in the middle of character strings will be excluded.

```
# This would be recognised as a comment in a job flow
print("# This would NOT be a comment.")  # This would be a comment
```

3.2.3 Execute the command

By writing the command you want to execute between **[ ]** these parenthesis as a character string, it becomes an **execution job** that can be executed as a command.

Note: If put inside `[]` parenthesis, the character string will be interpreted as a command. If the variable which is in the character string is substituted, it is possible to re-write what is in between those parenthesis to change the command. Variable substitution will be explained later in further detail.

The following is an example of how the command will be shown.

```
['whoami'] ->
print($RESULT)
```

If you run this job flow, the `whoami` command will be executed and as a result, the standard output will show in the console as a `print()` job. Usually it will show in the console as shown below.

```
[localhost] local: whoami

kompira
```

Note: Unless otherwise specified, as a result of executing the `whoami` command, the command will be executed on a host run by the job manager on the `kompira` account and it will display as `kompira`.

A character string beginning with `[localhost] local:` indicates which command was executed on which node. When the command is executed remotely, it will be displayed as `[<Host name>] run: <command>` or `[<IP Address>] run: <command>`.

3.2.4 $RESULT

`$RESULT` contains the execution result of the previous job. It is a special variable (status variable). The result of the command `whoami` will be stored in the character string "kompira".

Note: The format of the value stored in `$RESULT` depends on the type of job. For command jobs, the standard output is stored as a character string, but depending on the job it may be written numerically or alphabetically.

3.2.5 Linking jobs together

The arrow -> between jobs means that if the previous job was successful, subsequent jobs are to be executed. Therefore, you can run jobs in order by connecting jobs with a -> command.

If the job fails (even if the execution status of the command returns as anything other than 0), use the double arrow => to continue to the next process. The execution status of the previous command can be referred to by the `$STATUS` status variable.

The arrow linking these jobs is called *Connectors*, and there are 4 kinds in the job flow.

3.3 Use a variable

3.3.1 Variable definition

Variables can be defined using the syntax `{<variable definition> | <job>}`. `<Variable definition>` is written in the form `variable name = value (or expression)`. Multiple variable definitions can be described by separating them with a comma.

```
{ x = 'what do you get if you multiply six by nine?', y = 6 * 9 |
  print(x) -> print(y) }
```

In this case, the variable `x` is initialized with the string `'what do you get if you multiply six by nine?'` And the variable `y` is replaced with `6 * 9` It is initialized with the calculation result of the expression. You can write a job that refers to that variable, separated by a vertical bar `|` after the variable definition.

When you execute the above job flow, it will be displayed on the console as follows.

```
what do you get if you multiply six by nine?
54
```

3.3.2 Identifier

Characters written alphabetically as words or phrases in Unicode can be used for identifiers, variable names and so on. Japanese kanji, hiragana and English characters and underscores can be used (symbols other than underscores cannot be). However, you can not use the numbers `[0-9]` at the beginning of the identifier.

Therefore, the following character strings can be used as identifiers.

```
x, foo123, RESULT, __reserved_variable__
```

The following character strings can not be used as an identifier.

```
1st, foo-bar, @id, #hash
```

In addition, the following words are used as keywords and therefore cannot be used as variable names.

```
and             break           case            choice
continue        elif            else            false
for             fork            if              in
not             null            or              pfor
then            true            while
```

3.3.3 Scope

The valid range (scope) of the variable is the range enclosed in `{ }` parenthesis. Variables that are not defined within the scope can not be referenced, so the following job flow will result in an error during execution.

```
{ x = 'hello' |      # Scope of variable x is ...
    print(x) }       # ... up to here.
-> print(x)          # This is outside the scope.
```

It is possible to nest scopes as follows.

```
{ x = 'outer', y = 999 |
    print(x) -> print(y)
    -> { x = 'inner' |
        print(x) -> print(y) }
    -> print(x) -> print(y)
}
```

Execution of this job flow leads to the following, the scopes of `x = 'inner'` are the 3rd to 4th lines, and the 5th line reveals the outer scope.

```
outer
999
inner
999
outer
999
```

That is, the scoping rules of variables in the job flow are the same as in C and Java.

3.3.4 Assigning Variables

To change the value of the defined variable, substitute the variable as follows `[variable = value (or equation)]`.

```
{ x = 'outer', y = 'foo' |
    print(x) -> print(y) ->
    { x = '1st' |
        print(x)
        -> [x = '2nd'] -> print(x)
        -> [x = '3rd'] -> print(x)
        -> [y = 'bar']
        -> [z = 'baz'] }
    -> print(x) -> print(y) }
-> print(z)
```

When the scope is nested, the inner most scope is assigned to what was once the outer most scope, taking on its variable definition as well as its original location.

If you assign a value to an undefined variable, the variable is newly defined as the **outermost scope (job flow scope)** and set to that value. In the above example, since the variable `z` which is assigned a value in line 8 is undefined at that point, it is newly defined in the outermost scope and is displayed in line 10.

The outermost scope is not explicitly surrounded by `{}`, but you should imagine that there is a scope enclosing the entire job flow.

The execution result of the above job flow is as follows.

```
outer
foo
1st
2nd
3rd
outer
bar
baz
```

Note: Status variables such as `$RESULT` and `$STATUS` are internally set values by Kompira and as such, status variables cannot be assigned values in the job flow.

3.3.5 Array and Dictionary

Array

If you want to keep multiple values at once, use an array or dictionary. An array is described by separating multiple values or expressions with commas in square brackets as follows `[expression, ...]`. To access array elements, you can define them using square brackets using an index that starts with 0. You can also rewrite array elements with `[value >> array element]`.

```
[arr = [1, true, 'foo' ,['nested', 'array']]] ->
print(arr[1]) ->              # get array elements
[false >> arr[1]] ->          # set array elements
[arr = arr + ['added']] ->    # add array elements
print(arr[3][1]) ->           # get nested array elements
print(arr)                    # print() can print array
```

When this job flow is executed, it executes as follows.

```
true
array
[1, false, 'foo', ['nested', 'array'], 'added']
```

If a negative value is specified as an index, elements are accessed from the back of the array.

```
[arr = [1, true, 'foo']] -> print(arr[-1])
```

The execution result of this job flow is as follows.

```
foo
```

Dictionary

The dictionary describes `{identifier = expression, ...}` with a comma in the brackets delimiting multiple `identifier = value`. Access to dictionary elements is possible by specifying dot notation or by placing an identifier in square brackets. Also, rewriting dictionary elements is possible by changing it to `[value >> dictionary element]`.

```
[dic = {foo=1, bar=true, baz={a=123, b=456}}] ->
print(dic.foo) ->             # get dictionary elements (dot notation)
```

(continues on next page)

(continued from previous page)

```
[false >> dic.bar] ->          # set dictionary elements
print(dic['bar']) ->           # get dictionary elements (square bracket notation)
[[1,2,3] >> dic.arr] ->        # add dictionary elements
print(dic.baz.a) ->            # get nested dictionary elements
[777 >> dic.baz.a] ->          # set nested dictionary elements
[999 >> dic['baz']['b']] ->    # set nested dictionary elements
print(dic)                     # print() can print dictionary
```

When this job flow is executed, it executes as follows.

```
1
false
123
{foo=1, bar=false, baz={a=777, b=999}, arr=[1, 2, 3]}
```

3.3.6 Template character string

In the job flow, you can expand the value of a variable in a character string. If there is a placeholder consisting of `$` and an identifier in the string as shown below, that part can be replaced with the variable value indicated by the identifier.

```
[service = 'http', port = 80] ->
print('Port $port is used by $service')
```

When this job flow is executed, it executes as follows.

```
Port 80 is used by http
```

In place of `$identifier`, placeholders can also be written with the notation `${identifier}`. So when identifiers are not delimited in strings, please use `${identifier}` instead.

```
[w=640, h=480] ->
print("width=${w}px, height=${h}px")
```

You can also expand the value contained in the following dictionary by writing `%` after the string. In that case, write a placeholder of `%` and an identifier in the string.

```
print('Port %port is used by %service' % {service = 'http', port = 80})
```

The dictionary that follows `%` is ok to be a variable, such as shown below.

```
[ctx = {service = 'http', port = 80}] ->
print('Port %port is used by %service' % ctx)
```

In any of the notations, if the variable or dictionary element specified by the placeholder is undefined, it remains in the string as is, including `$` and `%`.

3.3.7 Parameters

The job flow can receive parameters at the time of execution.

You can define a variable as a parameter with the notation `|variable name|` enclosing the variable name at the beginning of the job flow with vertical bars. You can also define default values for parameters by writing `|variable name = value (or expression) |`. Please note that parameters that do not have default values need to specify values (can not be omitted) when executing the job flow.

In the following job flow, we define two parameters `command` and `wait`, and `wait` has a default value of 10.

```
| command |
| wait = 10 |

print('Execute the command "$command" after $wait seconds.') ->
["sleep $wait"] ->
[command] ->
print($RESULT)
```

Note: Parameters are evaluated in order from top to bottom, at the start of the job flow's execution. Therefore, you can also use expressions that refer to the values of the parameters that appeared earlier.

3.4 Remotely run commands

The next step is to try executing the command on a different host, from the host where the job manager is running next.

3.4.1 Specified by the control variable

Firstly, this is how to designate hosts and accounts to execute commands with control variables.

```
[__host__ = '<Hostname or IP-Address>',
 __user__ = '<Username>',
 __password__ = '<Password>']
-> ['hostname'] -> print($RESULT)
-> ['whoami'] -> print($RESULT)
-> ['echo Hello World'] -> print($RESULT)
```

Note: Please re-write your `<host name>`, `<user name>` and `<password>`.

`__host__`, `__user__` and `__password__` are the **reserved variables** in Kompira and these variables are the host name (or IP address), user name, and password. After setting the password , you can execute it using the host and user name you want to process subsequent remotes on.

If successful, the execution results should be displayed as follows

```
<Hostname>
<Username>
Hello World
```

If the host name is incorrect, or the user name or password are incorrect, the job flow fails and processing is aborted.

3.4.2 Node information and account information settings

If you create a node information object and an account information object on the Kompira file system, you can designate them from the job flow as the target server for command execution.

Suppose now that you create a node information object `test_node` and an account information object, `test_account` and that the host name, user name, and password information are set appropriately. Then, you can describe the command from the job flow in the same directory concisely as follows by using the control variable `__node__` for specifying the node information object, and `__account__` for specifying the account information object.

```
[__node__ = ./test_node, __account__ = ./test_account]
-> ['hostname'] -> print($RESULT)
-> ['whoami'] -> print($RESULT)
-> ['echo Hello World'] -> print($RESULT)
```

Note: Referencing to a Kompira object from the job flow can be done by describing it as relative path or absolute path. In the above example, we specify the objects in the same directory starting with `./`, but you can specify the path starting with `../` or `/` relative to the parent directory or root directory.

You can also omit the `__account__` specification if you have set a default account for `test_node`.

```
[__node__ = ./test_node]
-> ['hostname'] -> print($RESULT)
-> ['whoami'] -> print($RESULT)
```

In addition, you can specify the control variable as a parameter of the job flow, so you can also create a job flow that specifies the controlled node at run time.

```
|__node__ = ./test_node|
-> ['hostname'] -> print($RESULT)
```

3.4.3 Execution by sudo

If root privilege is required for command execution, set the control variable `__sudo__` to `true` and set the settings to sudo mode.

```
|__node__ = ./test_node|
-> ['whoami'] -> print($RESULT)
-> [__sudo__ = true]
-> ['whoami'] -> print($RESULT)
```

When this job flow is executed, it is displayed on the console as follows:

```
<Username>
root
```

Warning: In order to execute the command correctly in sudo mode, the user must be registered in the sudoers file. Otherwise, processing will fail (abort) when executing the remote command in sudo mode. For details, refer to the manual sudoers(5).

Note: When executing a command execution job that does not specify a host in sudo mode, you need to register the user of the server (usually the server on which Kompira is installed) that is running the job manager in the sudoers file. In addition, it is necessary to add a setting to invalidate the requiretty flag to the sudoers file as follows:

```
Defaults:kompira    !requiretty
```

3.5 Manipulating Jobs with Control Structures

3.5.1 Conditional branch

To branch processing according to the execution result of the previous job or the contents of the variable , use an `if` block or `case` block.

If block

If you use an `if` block, you can branch the process according to the result of the conditional expression.

```
['echo $$RANDOM'] ->
[x = int($RESULT)] ->
{ if x % 2 == 0 |
    then: print('$x is an even number')
    else: print('$x is an odd number')
}
```

In the above, the `then` clause is executed if the remainder of the variable `x` divided by 2 equals 0, otherwise the `else` clause is executed. `['Echo $$ RANDOM']` shows the environment variable `RANDOM` that returns a random number and `[x = int($RESULT)]` converts the result string to an integer `x`.

In addition to true / false, if you wish to further branch processing use the `elif` clause.

```
{ if x % 3 == 0 and x % 5 == 0 |
    then: print('FizzBuzz')
    elif x % 3 == 0: print('Fizz')
    elif x % 5 == 0: print('Buzz')
    else: print(x)
}
```

Alternatively, you can omit the `else` clause, or you can omit the `then` keyword.

```
[command] =>
{ if $STATUS != 0 | print('An error occurred: ' + $ERROR) }
```

In the example above, if you execute the command indicated by the contents of the `command` variable and the value of the `$STATUS` status is not 0, the `print` job will print the standard error output (`$ERROR`).

Case block

A conditional branch with a `case` block can be written as

```
['cat /etc/redhat-release'] ->
{ case $RESULT |
    'CentOS*release 7.*': print("This is CentOS")
    'Red Hat*release 7.*': print("This is Red Hat")
    else: print("CentOS/Red Hat 7.x is required")
}
```

In this example, a conditional branch is made to determine the type of the OS with the contents of the file `/etc/redhat-release`, and the pattern string may contain `*` or `?` for which Unix wired cards can be used.

The mapping of strings in the `case` block is done sequentially from the first pattern and only the job flow series following the first matched pattern is executed.

If none of the patterns match, the following will appear:

- If the `else` clause is included, the job flow sequence is executed.
- If the `else` clause is not included, the whole `case` block will fail(`$STATUS` is set to 1).

Note: Note that unlike the `if` block, the `else` clause is omitted in the `case` block, and the entire block fails if you do not match any of the conditions. If you do not do anything in the `case` block and not even an error occurs, try writing `else: []` and a skip job in the `else` clause.

3.5.2 Repetition

Repetition uses the `for` block or the `while` block.

for blocking

Some objects that Kompira can handle include complex data such as arrays and dictionaries or child elements such as directories. Use a `for` block if you want to perform the same processing on child elements (values and objects) contained in an object . The `for` block should be expressed as below

```
{ for <loop variable> in <object containing child elements> | job... }
```

For example, in the `in` clause, you can refer to the list of objects in the directory by writing a <directoryprefix> ::.

```
{ for t in /system/types | print(t) }
```

In this example, all the objects in the /system/types directory are referenced one by one with the loop variable `t` and are output to the console by the `print()` job. If you pass a Kompira object to the `print()` job, its absolute path is output to the console and the result is as follows.

```
/system/types/TypeObject
/system/types/Directory
/system/types/License
/system/types/Virtual
/system/types/Jobflow
/system/types/Channel
    :
```

In the `in` clause it is also possible to write a direct array as follows.

```
{ sum = 0 |
    { for i in [1,2,3,4,5,6,7,8,9,10] |
        [sum = sum + i]
    } ->
    print('The total of 1 to 10 is ${sum}.')
}
```

This job flow calculates and outputs the total value, from numbers 1 to 10.

```
The total of 1 to 10 is 55.
```

Also, if you write a dictionary following the `in` clause, you can refer to the list of identifiers contained in that dictionary sequentially.

```
[dic = {a=10, b=20, c=30}] ->
{ for k in dic |
    print("$k = %{$k}" % dic)
}
```

When this job flow is executed, it is displayed on the console as follows:

```
a = 10
b = 20
c = 30
```

Note: Before template expansion with `%` , In `%{$k}`, The `$k` part is replaced by the identifier of the dictionary. Therefore, each time it repeats, it expands to `%a`, `%b`, `%c` and that is the value of each element of the dictionary `dic`. The template is expanded and displayed as `10`, `20`, `30`.

While block

If you want to iterate through the job while satisfying certain conditions, use the `while` block instead. The syntax of the `while` block is as follows.

```
{ while <expression> | job... }
```

For example, an "Euclidean algorithm" which finds the greatest common divisor of two given numbers, is an algorithm that iterates until the remainder becomes 0, but when it is described using a `while` block, shows as follows

```
|x = 165|
|y = 105|
[m = x, n = y] ->
{ while n != 0 |
    [r = m % n] ->
    print("The remainder of $m and $n is $r.") ->
    [m = n, n = r]
} ->
print("The greatest common divisor of $x and $y is $m.")
```

In this `while` block, `n` is `n` while `n` is not 0, `n` for `m`, `m` for `n` and the job of substituting (and displaying) the remainder of `n` is repeated. When executed, it displays as follows.

```
The remainder of 165 and 105 is 60.
The remainder of 105 and 60 is 45.
The remainder of 60 and 45 is 15.
The remainder of 45 and 15 is 0.
The greatest common divisor of 165 and 105 is 15.
```

3.5.3 Calling a job

Calling a job flow

To call another job flow from one job flow, use the following syntax.

```
[<Jobflow object>]
```

Here is an example of creating a job called "sub job" and calling it. First, define the sub job as follows, under the appropriate directory.

```
print("This is subjob.") ->
return("Succeeded.")
```

The `return` job terminates the sub job and returns the result to the calling job.

Next, create a main job that calls this sub job under the same directory.

```
print("Call the subjob.")
-> [./SubJob]                  # call the subjob
-> print($RESULT)              # return the result of subjob
```

Note that "./" is added to the beginning of the string at the point of specifying the call of the "sub job". This indicates that "sub job" is defined in the same directory as the directory in which the current job flow is defined.

The execution result of the sub job can be received as `$RESULT`. When the above main job is executed, it is displayed as follows.

```
Call the subjob.
This is subjob.
Succeeded.
```

Passing parameters to job flows

When invoking a job flow, you can also pass parameters using the following syntax:

```
[<Jobflow object> : <parameter list> ... ]
```

First, extend the sub job and add the parameters as follows.

```
|parameter1 = 'Hello'|
|parameter2 = 'World'|

print("This is subjob.")
-> print(parameter1)
-> print(parameter2)
-> return("Succeeded.")
```

In this state, if you execute the main job as it is, the following will be displayed. Since no parameters are specified at the time of invocation, you can see that the default parameters defined on the sub job side are used.

```
Call the subjob.
This is subjob.
Hello
World
Succeeded.
```

To pass parameters to this sub job and call it, extend the main job as follows. When calling a sub job, you can write values following `:` so that you can pass them to the sub job as parameter values.

```
print("Call the subjob with parameter.")
-> [./SubJob: 'HELLO', 'WORLD']
-> print($RESULT)
```

When you do this, the result is as follows.

```
Call the subjob with parameter.
This is subjob.
HELLO
WORLD
Succeeded.
```

It is also possible to specify the parameter name defined on the side of the called job flow and pass the parameter value. This is convenient when you want to specify only some parameters.

```
[./SubJob: parameter2='WORLD']
```

Note: Please be aware that specifying a parameter name that is not defined on the called side, or trying to pass more parameter values than defined, will result in an error.

Execution of a script job

If you want to create more complicated jobs, it would be better to combine existing scripting languages such as bash, perl, ruby, python, etc. than to use the Kompira job flow language. By creating a script job on the Kompira file system, it becomes possible to call these script language programs from the job flow.

Let's look at an example of writing a script job using a shell script and calling it from a job flow. First, save a simple shell script as shown below as a script job.

```
#! /bin/sh
echo Hello world from shell script
```

For scripts to be executed in a Unix environment, please describe the shebang line beginning with `#!` on the first line appropriately. For scripts running on a Windows environment, you will need to specify the extension (eg bat/vbs/ps1) appropriately.

If you save this script job as "sample_script", the job flow for executing this script job is as follows.

```
print('Execute the ScriptJob') ->
[./SampleScript] ->
print($RESULT)
```

If you do not specify __node__ or __host__, this script will be executed after being transferred to the machine on which the job manager is running. The output of the execution result is stored in $RESULT like the execution result of the remote command.

Note: The script is transferred to the host specified at runtime as a temporary file and deleted after it is executed.

Parameters can also be passed to script jobs. The script job side receives parameters as command line arguments.

On the caller side of the script, pass parameters as keyword-less arguments as follows.

```
[./SampleScript: 'parameter1', 'parameter2']
```

3.6 Manipulating objects

3.6.1 Referencing objects

Information handled by Kompira, such as job flow and environment variable definitions, are managed as a **Kompira object** in a unified manner on the Kompira file system. These objects can then be accessed from the job flow by specifying a path like a Unix file system.

In previous examples, references to Kompira objects were specified by relative paths. In this case, the path of the object is specified based on the directory in which the job flow being executed is defined.

For example, if the running job is /some/path/jobflow, referencing the object with the relative path /subdir/object will result in /some/path/subdir/object being accessed.

Also, referring to the relative path ../object will access /some/object.Relative paths starting with ../ refer to objects in parent directories.

Of course you can also reference the object directly as an absolute path like /some/path/object.

Warning: Do not forget to add ./, ../ or / to the beginning of Kompira object references. Kompira recognizes a character string beginning with ./, ../ or / as a **path identification**, otherwise it recognizes it as an identifier of a variable.

If you want to concatenate paths and reference objects, use the path() built-in function. For example, when preparing a job flow for "resource information acquisition" for each type of node, if you want to execute the job flow by designating the node and the node type, you can refer to the job flow by assembling the path.

```
|node|
|node_type = 'Linux'|
|job_name = 'GetResourceInfo'|
[job = path(./DefinitionsByNodeType, node_type, job_name)] ->
[job: node]
```

Here, if the default argument is passed to the path() function as it is, refer to the job flow named ./node definition/Linux/resource information acquisition by using the variable job and node as a parameter.

3.6.2 Browse and update properties

Each Kompira object has a "property" defined in the system. For example, properties are the name and path of the object, creation date and time, and so on. For details on the properties of Kompira objects, see *Properties*.

To refer to the properties of a Kompira object, use the dot notation `object.property name`. In the following job flow, it lists the Kompira objects in the directory which was the parameter `dir` and its properties 'Owner (owner)', 'Updated date (update)', 'Type name (type_name)', 'Display name (display_name)' is displayed by dot notation.

```
| dir = / |
{ for obj in dir |
    [attr = {
        owner = obj.owner,
        updated = obj.updated,
        type = obj.type_name,
        name = obj.display_name
    }] ->
    print("%owner %updated <%type> %name" % attr)
}
```

To update the property value of a Kompira object, use the output job `[value >> object.properties]`.

```
["Description of the Object" >> obj.description]
```

Note: However, please note that some of the properties can not be updated from the job flow and some are not writable. See *Properties* for details.

3.6.3 Referencing and updating fields

Each Kompira object has a "field" defined for each type. You can see what fields are defined for each type defined in the system by looking at the definition information of each type under `/system/types/`.

Fields of Kompira objects can be referenced by `object['field_name']` or `object.field_name`.

Note: Please note that the property of the object can also be accessed by dot notation. You can also define a field with the same name as the property, but dot notation refers to the property value in preference.

For example, in the node information object, fields such as "host name (hostname)" and "IP address (ipaddr)" are defined. To refer to these values in the job flow, you write as follows.

```
|node = ./node|
print(node['hostname'], node.ipaddr)
```

Since values of fields can be referenced like dictionaries, template expansion with `%` is also possible.

```
|node = ./node|
print('%hostname: %ipaddr' % node)
```

Also, to update the field value of the Kompira object, use the notation `[value >> object['field_name']]` or `[value >> object.field_name]` in the output job. For example, to update the "Wiki text" field ('wikitext') of Wiki page type, write as follows.

```
['= Sample Wiki\n' >> ./wiki['wikitext']]
```

You can also write the result of an expression in an output job, so you can modify the referenced field value and rewrite it as follows.

```
|wiki = ./wiki|
|types = /system/types|
["= Type list\n" >> wiki.wikitext] ->
{ for type in types |
    [wiki.wikitext + "* $type: (" + type.description + ")\n" >> wiki.wikitext]
}
```

In the example above, we create a wiki page that lists the system standard type objects in `/system/types`, listing their paths and descriptions.

3.6.4 Calling methods

Some Kompira objects have methods. To invoke a method on an object we use the following syntax:

```
[ <object> . <method name> : <parameter list> ... ]
```

For example, to add an object, the directory type object has a method called `add`. The `add` method is called by specifying three parameters `name`, `type_obj`, and `data`. In the following example, create an environment variable type (/system/types/Environment) object with the name 'ENV' in the same directory as the job flow and put the `{k1 = 'value1', k2 = 'value2'}` as given.

```
[./.add: 'ENV', /system/types/Environment, {
    environment={k1='value1', k2='value2'}
}]
```

Here, the relative path identification `./` refers to the Kompira object indicating the directory to where this job flow resides. You can also pass object references as variables, so you can write:

```
[dir = ./, type=/system/types/Environment] ->
[dir.add: 'ENV', type, {environment={k1='value1', k2='value2'}}]
```

In the parameter string, you can pass a value by specifying the parameter name.

```
[dir = ./, type=/system/types/Environment] ->
[dir.add: 'ENV', type_obj=type, data={environment={k1='value1', k2='value2'}}]
```

3.7 Waiting for an event

Job synchronization and event waiting processing can be described in the job flow using the channel.

3.7.1 Transmission of messages

To send a message to the created channel, use the `send` method. A new channel can be created as "/home/guest/test channel".

The job flow for sending a message to a channel is as follows.

```
[/home/guest/TestChannel.send: 'Hello']
-> print('Sent a message.')
```

Next, define the job flow to receive messages from the channel as follows.

```
</home/guest/TestChannel>
-> [mesg = $RESULT]
-> print('Message "$mesg" was received.')
```

Please execute the above job flow. It is deemed successful if a message is output to the process console of the job flow execution on the receiving side as follows.

```
Message "Hello" was received.
```

If you execute the sender job flow more than once, messages will be accumulated on that channel by that amount. Every time the receiver's job flow is executed, it extracts one message from that channel and outputs it. If the message on the channel is empty, the receiving job flow waits until a new message arrives.

Note: Using the `kompira_sendevt` command you can send arbitrary information to the channel from an external system. For example, by transmitting alert information from the monitoring system to the channel, it is possible to process the procedure at the time of failure by the job flow. For information on how to use the `kompira_sendevt` command, see *Coordination with other systems*.

3.7.2 About event jobs

A job enclosed by < and > is called an event job. Event jobs can be used in combination in the job flow in the same way as other jobs.

The format of an event job is as follows.

```
< <object> : <parameter list> ... >
```

For object names, specify objects of channel type (and similar type: mail channel type etc). Other events that specify an object that can not be queued, result in a runtime error.

3.7.3 Specify a timeout for message retreival

Wait for the arrival of the message from the channel, specify the parameter `timeout` for the event job to time out and continue the process if it does not arrive within the fixed time.

```
print('Wait for a message from channel.')
-> <./TestChannel: timeout=10>
=> { if $STATUS==0 |
    then: [mesg=$RESULT]
          -> print('Message "$mesg" was received.')
    else: print('Timeout occurred.') }
```

Note that if the message does not arrive within the number of seconds specified by `timeout`, the event job will fail, so be aware that => binds the next job.

Note: If the channel is deleted while waiting for arrival of the message, the event job fails and sets $ STATUS to -1.

3.7.4 Selective reception from multiple channels

It is also possible to wait for the arrival of messages from multiple channels by using the `choice` block. In this case, the processing of the channel on which the message arrived earlier continues.

Example of choice block usage:

```
print('Wait for message from channel1 and channel2.')
-> { choice |
    <./Channel1> -> [mesg=$RESULT]
        -> print('Message "$mesg" was received from Channel1.')
    <./Channel2> -> [mesg=$RESULT]
        -> print('Message "$mesg" was received from Channel2.')
}
-> print('OK')
```

3.8 Access externally

3.8.1 Send mail

To send the mail, use the built-in `mailto` job.

```
[subject = 'Test mail',
 body  = 'Send a test mail.']
-> mailto(to='taro@example.com', from_user='hanako@example.com',
          subject=subject, body=body)
-> print('Sent a mail.')
```

Arguments of the `mailto` job include `to` (destination mail address), `from_user` (source mail address), `subject` (mail title), `body` (mail body text).

When sending mail to multiple addresses, pass the list of mail address strings to the `to` argument as follows.

```
mailto(to=['taro@example.com', 'jiro@example.com'], from_user='hanako@example.com',
       subject=subject, body=body)
```

Warning: We use the sendmail command to send mail. If the mail can not be sent successfully, check the settings of the sendmail command and check whether the mail can be sent correctly with the sendmail command from the Kompira server.

3.8.2 HTTP Access

For HTTP access to web servers etc., use the built-in `urlopen` job. Simply passing URL only to `urlopen()` will result in GET access.

```
|url = 'http://www.kompira.jp'|
urlopen(url)
=> [status = $STATUS, result = $RESULT]
-> { if status != 0 |
then:
    print('HTTP access failed.')
elif result.code != 200:
    print('HTTP status code is %code.' % result)
else:
    print(result.body)
}
```

The result of successful access with `urlopen()` is returned in the dictionary. `code` contains the HTTP status code, and `body` contains the contents of the response.

As Kompira at version 1.5.0 does not have the function to parse HTML, we create a simple script job like the following, `html_parse`. This script extracts the part specified by the parameter as text from the HTML passed to standard input.

```
#! /usr/bin/python
import sys;
from lxml import html;
if __name__ == '__main__':
    doc = html.fromstring(sys.stdin.read().decode("utf-8"))
    for e in doc.xpath(sys.argv[1]):
        print html.tostring(e, method="text", encoding="utf-8")
```

`urlopen()` can also pass POST access by passing dictionary data to parameter `data`. As an example, consider a job flow of checking the product vendor from the first half (OUI) of the MAC address assigned to the network interface. OUI is managed by the organization called IEEE, (see weblink) http://standards.ieee.org/develop/regauth/oui/public.html. There is a form on this page, so that OUI is entered in the entry field named `x`. Also, since CGI called `/cgi-bin/ouisearch` is executed when searching, you need to POST access to pass OUI as data `x` to that CGI as data .

```
|oui = '00-00-00'|
urlopen('http://standards.ieee.org/cgi-bin/ouisearch', data={x=oui})
-> [./html_parse << $RESULT.body: '//pre']
-> print($RESULT)
```

Since there is a result in the <pre> tag on the search result page, we pass the parameter `//pre` to the `html_parse` script to extract it. This parameter is specified by the syntax for specifying part of an XML document called XPath.

By executing this job flow, you can see that vendor information can be acquired from an external web page as follows.

```
[localhost] local: (/tmp/tmpxaL7DG //pre) < /tmp/tmpktOTMU

  OUI/MA-L                    Organization
  company_id                  Organization
                                  Address


  00-00-00    (hex)           XEROX CORPORATION
  000000      (base 16)       XEROX CORPORATION
                              M/S 105-50C
```

(continues on next page)

(continued from previous page)

```
                                    800 PHILLIPS ROAD
                                    WEBSTER NY 14580
                                    UNITED STATES
```

3.9 Controlling processes

When a job flow is executed, it is managed in Kompira as a process execution unit until the end of the job flow, and the process will sequentially execute the jobs described in the job flow sequentially.

3.9.1 Process Termination

If there are no more jobs to continue, such as reaching the end of the job flow, or if you join the job using `->` when the executed command fails, the process will automatically finish.

Otherwise, you can use an `exit` job or `abort` job to explicitly terminate a running process.

exit

To terminate a running process, use the built-in `exit` job. If you call `exit()` without specifying arguments, the process terminates normally immediately.

```
exit()
```

You can also specify the exit status code with the argument of the `exit` job. In the following example, after executing the command specified by the parameter `command`, the standard error output and standard output are displayed regardless of the result (success / failure), and then the command execution result is processed as a status code.

```
|command|
[command]
=> [status=$STATUS, stderr=$ERROR, stdout=$RESULT]
-> { if stderr | print(stderr) }
-> { if stdout | print(stdout) }
-> exit(status)
```

Please note the difference between `exit` and `return`. For example, calling the `exit` job at a sub job called from the main job will terminate the running process (control will not be returned to the main job and will end immediately). On the other hand, if you call the `return` job at a sub job, control is returned to the main job rather than terminating the process, and processing continues immediately after the execution job that invoked the sub job.

However, if the caller does not exist, for example, if you call the `return` job from the job flow you pressed by pressing the "execute button" directly, the job will be terminated at that point and the process will terminate.

abort

You can abnormally end a running process by calling the built-in `abort` job, for example, when the job can not be continued. In the example below, when accessing the URL specified by the parameter with `urlopen`, if the HTTP access fails or if the HTTP status code is not 200, abnormally terminate (abort) the process.

```
|url|
urlopen(url)
=> [result = $RESULT, status = $STATUS]
-> { if status != 0 | abort('HTTP access failed.') }
-> { if result.code != 200 | abort('HTTP status code is %code.' % result) }
-> return(result.body)
```

The `abort()` job is almost identical to `exit(status=1)` because it automatically terminates the process with an exit status code set to 1.

3.9.2 Child Process Activation

Kompira's concurrent behavior of multiple processes can be used in a job flow by starting a "child process".

A child process is a copy of a parent process (that is, a process that started a child process) at the time of activation, local variables and special variables have the same value, but since it can not share or reference between processes, Please note that it is not possible to rewrite the variables of the parent process from the process (the same is true for the reverse direction).

fork

It is possible to start multiple child processes at once using the `fork` block. Below is an example of a job flow that causes the execution results of the sub job "processing A" to be processed in parallel with the sub jobs "processing B" and "processing C", respectively.

```
[./ProcessA] -> [result = $RESULT] ->
{ fork |
    [./ProcessB: result] -> print('ProcessB is finished.')
    [./ProcessC: result] -> print('ProcessC is finished.')
} -> print('All child processes have terminated.')
```

There are places where jobs are not connected by connectors in the `fork` block, but this is a "job flow expression" delimiter, and in the above example they are two job flow expressions. When these two job flow expression parts operate in parallel as child processes and their execution is completed, the job of the parent process continues to be output to the console "all the child processes have ended" .

When starting a child process in the job flow, the child process started is displayed on the "child process list" tab of the process details screen of that process. Conversely, please be aware that child processes are not displayed on the "process list" screen.

pfor

By using the `pfor` block instead of the `for` block, iterations can be executed as a parallel process all at once.

For example, if you want to manage the managed nodes in the "node list" and want to execute the same job "configuration information collection" on all managed nodes, you can write as follows using the `for` block (Assume that the configuration information collection specifies the node to be processed with parameters).

```
|job = ./CollectConfigurationInformation|
{ for node in ./NodeList |
    [job: node]
} -> print("Processing of all nodes has ended.")
```

If this "configuration information gathering" job is submitting a command which takes time to process to the remote node, this process is "waiting" more often. As a result, the load is low, but it will take a long time to finish processing for all nodes.

If you use `pfor` instead of `for` then you will invoke the child process on each node and execute the 'gather configuration information' job for that child process. Then, processing can be executed in parallel by another node even if it is in the "waiting state" due to the processing on a certain node, so it is possible to shorten the processing time by increasing the job execution efficiency as a whole.

```
|job = ./CollectConfigurationInformation|
{ pfor node in ./NodeList |
    [job: node]
} -> print("Processing of all nodes has ended.")
```

3.9.3 Detaching from the parent process

The parent process that started the child process using `fork` or `pfor` will wait for all child processes to finish, so the parent process will not be able to run the new job during that time. However, there are cases where you want to continue processing on the parent process side without waiting for the child process to finish. In that case we can deal with it by detaching it from the parent process using `detach()`.

detach

For example, you often want to execute the same job flow each time you receive a message from a channel. In the following, every time a message is received, it is called by passing a message as a parameter to the job flow "message processing".

```
|chan = /system/channels/Alert|
|proc = ./MessageProcessing|
{ while true |
    <chan>
    -> [msg = $RESULT]
    -> [proc: msg]
}
```

If there is no relevance between multiple messages received from the channel, by simultaneously executing the message processing, when the messages arrive consecutively, the processing efficiency of the whole can be improved. To that end, it is necessary to operate the job flow that receives the message and "message processing" as a separate process. This is "message processing" in a child process using *"fork* `.

```
|chan = /system/channels/Alert|
|proc = ./MessageProcessing|
{ while true |
    <chan>
    -> [msg = $RESULT]
    -> { fork | [proc: msg] }
}
```

However, since the parent process waits until the "message processing" job is completed, even if a new message arrives during message processing, it can not be processed at the same time. So we use a `detach()` built-in job on the child process to separate the child process from the parent process.

```
|chan = /system/channels/Alert|
|proc = ./MessageProcessing|
{ while true |
    <chan>
    -> [msg = $RESULT]
    -> { fork | detach() -> [proc: msg] }
}
```

By detaching child process using `detach()`, the parent process will have no child processes to wait for processing completion, so the next job can be continued at that point. That is, the next message is received from the channel, and a new "message processing" can be activated even if "Message processing" started earlier is not completed yet.

The child process becomes a normal process instead of a child process by using `detach()` and it will be displayed on the 'Process List' screen instead of the 'Child Process List' of the parent process.

By combining `fork` and `pfor` with `detach()`, you can easily write somewhat complicated parallel processing in this way.

CHAPTER

FOUR

KOMPIRA JOBFLOW LANGUAGE REFERENCE

Author Kompira development team

4.1 Introduction

This document explains the tokens and syntax and meanings of the job flow language. A description of built-in functions and embedded jobs can be found in *Kompira Standard Library*.

4.1.1 Syntax Notation

In this document, the syntax is shown using extended BNF. Extended BNF uses symbols such as "*" representing zero or more repetitions, "+" representing one or more repetitions, "?" Representing an optional element, in addition to normal BNF. The parentheses "(" and ")" are also used to group multiple elements together.

4.2 Lexical structure

This chapter specifies the lexical structure of the job flow language. The program text of the job flow language is written in Unicode. The text is delimited by vocabulary units called tokens by Kompira's lexical analyzer.

4.2.1 Comment

Comments begin with a hash character (#) that is not included in a string literal, and end at the end of the line. Comments are skipped by the lexical analyzer.

4.2.2 Blanks

Newline characters, spaces, tabs, and form feeds are treated as blanks. Whitespace is skipped by the lexical analyzer.

4.2.3 Identifiers

The identifier (IDENTIFIER) is defined by the following regular expression.

```
IDENTIFIER = [^\W0-9]\w*
```

\w matches any Unicode word character. This includes letters, numbers, and underscores that can be part of a word in any language. \W means [^w]. The length of the identifier is unlimited. Identifiers are case-sensitive.

Keywords

The following character sequences are reserved as keywords and can not be used as identifiers.

```
and             break           case            choice
continue        elif            else            false
for             fork            if              in
not             null            or              pfor
session         then            true            try
while
```

Reserved identifier Class

Identifiers of the form __*__ are reserved in the system for control variables and have special meanings. Since it is possible that unexpected behavior of the job flow may occur, it is better to avoid users using these names as identifiers.

Special identifiers

An identifier starting with $ is a special identifier used for special variables as defined below.

```
SPECIAL_IDENTIFIER = "$" IDENTIFIER
```

4.2.4 Object Path

The object path points to the location of the object on the Kompira file system. It is defined as follows:

```
      OBJECT_PATH = RELATIVE_PATH
                  | ("/" | RELATIVE_PATH) PATH_ELEMENT* LAST_PATH_ELEMENT
    RELATIVE_PATH = "./" | "../"
     PATH_ELEMENT = RELATIVE_PATH | IDENTIFIER "/"
LAST_PATH_ELEMENT = RELATIVE_PATH | IDENTIFIER
```

Note: Although a single "/" is also treated as an object path, it is not included in OBJECT_PATH as a token, because it is indistinguishable from the division operator "/" in terms of lexical analysis.

4.2.5 Literal

A literal is a source code representation of values of type String, Binary, Integer, Float, Boolean, Null, and Pattern.

String literal (STRING)

A string literal consists of zero or more characters enclosed in single quotation marks (‘) or double quotation marks (“)

```
""                  # empty string
'"'                 # "
'\''                # '
"This is a string"  # String containing 16 characters.
```

It can also be surrounded by corresponding triple quotes or double quotes. In this case, you can write non-escaped newlines and quotes.

```
'''''''             # '
"""String containing
line feed code"""   # String containing line feed code
```

Within a string literal, you can use escape sequences to represent certain non-expressable characters, such as newline characters and tab characters.

The list of escape sequences is shown below.

Escape sequence	Meaning
\	backslash()
'	Single quotations (‘)
"	double quotations(“)
a	ASCII terminal bell (BEL)
b	ASCII backspace (BS)
f	ASCII form feed (FF)
n	ASCII line feed (LF)
r	ASCII return (CR)
t	ASCII horizontal tab (TAB)
v	ASCII vertical tab (VT)
ooo	Characters with octal *ooo*
xhh	Characters with hexadecimal value *hh*

Binary literal (BINARY)

A binary literal consists of zero or more characters enclosed in single quotation marks (‘) or double quotation marks (“) with the prefix sign b.

```
b""                 # empty binary
b'"'                # "
b'\''               # '
b"This is a string" # binary containing 16 bytes.
```

Unlike string literals, you cannot use multi-line descriptions with a corresponding triplet of quotation marks. Nor can you include non-ASCII characters such as Japanese. Escape sequences can be used in the same way as string literals.

Integer literal (INTEGER)

Integer literals can be expressed in decimal. It is described by the following lexical definition.

```
INTEGER       = NONZERO_DIGIT DIGIT* | "0"
DIGIT         = [0-9]
NONZERO_DIGIT = [1-9]
```

Floating-point literal (FLOAT)

A floating-point literal is a representation of a floating-point number consisting of mantissa and exponential parts consisting of integer and decimal parts. It is described by a lexical definition.

```
FLOAT         = DIGIT+ "." DIGIT* ( [eE] [+-]? DIGIT+ )?
DIGIT         = [0-9]
```

An example is shown below:

```
3.1415926
0.5e-3          # 0.0005
12.3e+2         # 1230.0
9.E5            # 900000.0
```

Boolean literal (BOOLEAN)

A boolean literal has two boolean expressions of true (true) and false (false).

```
BOOLEAN = "true" | "false"
```

Null literal (NULL)

A null literal is a value indicating that there is no value, and it is written as null.

```
NULL = "null"
```

Pattern literal (PATTERN)

After the characters a pattern literal makes up, (‘e’ ‘g’ or ‘r’ making a pattern), more than 0 characters are surrounded with a single quotation mark (‘) or double quotation marks (“) (pattern string). The ‘i’ which indicates a mode at the end is sometimes added as an option.

```
r"(From|Subject): "      # regular expression pattern
g'*.txt'                 # glob pattern
e'kompira'i              # case-insensitive exact match pattern
r"windows(95|nt|2000)"i  # case-insensitive regular expression pattern
```

Within a pattern character string, the escape sequence is invalid and it is handled as it is. String substitution by ${identifier} is valid.

4.2.6 Symbols

Symbols are classified as operator symbol (OPERATOR), connector symbol (COMBINATOR), and delimiter.

Operators

The following tokens are operators:

```
+       -       *       /       %
<       >       <=      >=      ==      !=      =~      !~
```

Connectors

The following tokens are connectors:

```
->      =>      ->>     =>>
```

Delimiter

The following tokens are delimiters:

```
(   )   {   }   [   ]   |   ,   .   =   >>   <<   ?   ??
```

4.3 Value and type

Kompira's job flow language can handle various values (data) such as integers, character strings and dates.

4.3.1 Primitive types

Primitive type is a generic name of basic data types of Kompira's job flow language, which are four types: integer type, character type, boolean type, and null type. Values of primitive types are never shared with values of other primitive types.

Integer type (Integer)

The integer type handles types of values representing integers such as 0, 1, 1000, -9999. The integer type of Kompira is not limited in scope (as far as memory allows).

Note: Since the range of the Integer type field of the Kompira object is limited, writing data outside the range from the job flow to the Integer type field will result in a runtime error.

String type (String)

String type is a type for string values like “kompira” or “today is sunny”. Each element of the string is a letter. There is no character type in Kompira’s job flow language. A single character is represented as a string with only one element. Each character is expressed internally as Unicode.

If you write the value of a string representing an integer, such as “123” or “-999”, into an integer field, it will be implicitly converted to the corresponding integer value.

When the value of the string type is converted to Boolean type, the empty string (“”) corresponds to false, and the other string corresponds to true. Therefore, be aware that the string “false” corresponds to Boolean true.

The string type data has the following methods:

format (*args, **kwargs) Perform a string formatting operation. The string on which this method is called can contain literal text or replacement fields delimited by braces {}. Each replacement field contains either the numeric index of a positional argument, or the name of a keyword argument. Returns a copy of the string where each replacement field is replaced with the string value of the corresponding argument.

join (list) Return a string which is the concatenation of the strings in list. The separator between elements is the string providing this method.

find (sub[, start[, end]]) Return the lowest index in the string where substring sub is found within the slice s[start:end]. Optional arguments start and end are interpreted as in slice notation. Return -1 if sub is not found.

rfind (sub[, start[, end]]) Return the highest index in the string where substring sub is found, such that sub is contained within s[start:end]. Optional arguments start and end are interpreted as in slice notation. Return -1 on failure.

startswith (prefix[, start[, end]]) Return true if string starts with the prefix, otherwise return false. prefix can also be a tuple of prefixes to look for. With optional start, test string beginning at that position. With optional end, stop comparing string at that position.

encode ([encoding]) Converts the string to a byte string encoded with the specified encoding. If encoding is not specified, the string is encoded as ‘utf-8’.

endswith (prefix[, start[, end]]) Return true if the string ends with the specified suffix, otherwise return false. suffix can also be a tuple of suffixes to look for. With optional start, test beginning at that position. With optional end, stop comparing at that position.

lower () Return a copy of the string with all the cased characters converted to lowercase.

upper () Return a copy of the string with all the cased characters converted to uppercase.

replace (old, new[, count]) Return a copy of the string with all occurrences of substring old replaced by new. If the optional argument count is given, only the first count occurrences are replaced.

split ([sep[, maxsplit]]) Return a list of the words in the string, using sep as the delimiter string. If maxsplit is given, at most maxsplit splits are done.

rsplit ([sep[, maxsplit]]) Return a list of the words in the string, using sep as the delimiter string. If maxsplit is given, at most maxsplit splits are done, the rightmost ones.

splitlines ([keepends]) Return a list of the lines in the string, breaking at line boundaries.

strip ([chars]) Return a copy of the string with the leading and trailing characters removed. The chars argument is a string specifying the set of characters to be removed.

Binary type (Binary)

A binary type is a type whose value is a sequence of bytes. A sequence of bytes is similar to a string of characters, but the units of a sequence of bytes are not characters, but rather byte values (8-bit integers between 0 and 255).

The binary type data has the following methods:

decode ([encoding]) A byte sequence is interpreted with the encoding specified by the encoding and converted into a string. If the encoding is omitted, it will be interpreted as 'utf-8'.

hex () Converts each byte value in a string of bytes into a string of 2-digit hexadecimal notations.

Floating-point type (Float)

A floating-point type is a type that takes a floating point number as a value.

Boolean (Boolean)

Boolean type is a type that takes two values of truth - 1. value true (true) and 2. false (false)

Null type (Null)

A null type is a type that has only null values.

Pattern type (Pattern)

A pattern type is a type of a value that represents a pattern for matching with a character string. There are three types of patterns: 'r' (regular expression pattern), 'g' (glob pattern), 'e' (perfect match pattern). Also, you can combine capitalized and lowercase non-discriminating mode ('i') as pattern matching mode.

The regular expression pattern conforms to the regular expression of the re module of the programming language Python.

In the glob pattern, you can use Unix shell-style wildcards and correspond to the following special characters.

Pattern	Meaning
*	Matches everything
?	Matches any single character
[seq]	Matches any character in seq
[!seq]	Matches any character not in seq

An exact match pattern is simply a comparison of character strings.

Pattern type data has the following methods:

match (s) Attempts to match the string s with the pattern. If a match is true, or if the pattern is a regular expression pattern, it returns a dictionary containing matched information. If it does not match, it returns false.

The dictionary data returned when the regular expression pattern matches includes the following entries

Key	Meaning
group	String matched with regular expression
groups	List containing strings of all subgroups
groupdict	Named group dictionary
start	Start position of match
end	End position of match

4.3.2 Complex data type

A complex data type is a generic name of data types that can hold multiple elements of other types, and there are two types: array type and dictionary type.

Array type (Array)

Array type data is a data structure in which elements are arranged in one dimension, and elements can be accessed with an integer index. If the length of the array is n, the index is 0, 1, . . . , n - 1. Element i of array can be referenced by a[i]. If index i is negative, a[i] refers to element n + i.

If you access elements outside the range of the array, a run-time error occurs. Also, arrays can not be extended.

The array type data has the following methods:

add_item (value) Add data value to the end of array a.

del_item (index) Delete the element a[index] of the array a.

pop_item ([index]) Delete the element a[index] of the array a. If index is not specified, the last element is deleted.

Dictionary type (Dictionary)

Dictionary type data is a data structure that allows elements to access elements associated with any type of key except for complex data types. Elements associated with key k of dictionary d can be referenced by d[k]. If the key k is a string type value and it is a character string satisfying the lexical requirement of the identifier (IDENTIFIER), it can be referred to as d.k.

If you attempt to refer to an element with a key that is not included in the dictionary, it will result in an execution error. You can add new keys and elements by writing job (described later).

Dictionary type data has the following methods:

del_item (key) Delete element d[key] of dictionary d.

get_item (key[, default]) Get the element d[key] of dictionary d. If element d[key] does not exist, it returns default. The default value of default is null.

pop_item (key[, default]) Delete element d[key] of dictionary d. If element d[key] does not exist, it returns default. If default is not given and the element d[key] does not exist, an error occurs.

get_keys () Return a copy of the dictionary d list of keys.

4.3.3 Opaque data type

The opaque data type is a generic term for data types whose internal structure of data is hidden. Also, since it does not have a corresponding data constructor, it can not directly generate data by notation in the source code of the job flow program like a complex data type.

Object type (Object)

The value of the object type represents a reference to the object on the Kompira file system. The string representation of the value of the object type is the absolute path of that object. The property p of the object o can be accessed with op, and the field f can be accessed with the notation o[f]. If there is no property name or method name with the same name as the field f, the field can be referenced with the notation of o.f.

Kompira's object have fields and methods defined by a type object (TypeObject). For details, refer to the Kompira Object Reference (*Kompira Standard Library*)

File type (File)

The value of the file type represents the file data attached to the object with the file type field.

The following fields are defined in the file type value.

Field name	
name	Attachment (file) name
data	Attached file data
path	Local file path on the Kompira server (read only)
size	Data size (read only)
url	Download URL (read only)

Date and time data (Datetime)

A datetime value represents data that contains both the date and the time.

Date-time type values have the following read-only properties:

Key	Meaning
year	Year
month	Month (values from 1 to 12)
day	Day (value from 1 to the number of days in the given month and year)
hour	Time (value from 0 to 23)
minute	Minute (value from 0 to 59)
second	Seconds (values from 0 to 59)
weekday	With Monday as 0, Sunday as 6, a value representing the day of the week as an integer
date	Date data
time	Time data

Datetime data has the following methods:

format (dt_fmt) Converts the date / time data to a character string in the format specified by dt_fmt. Format specification of this format conforms to C language strftime() function.

An example is shown below:

```
[dt = now()] -> print(dt.format('%Y-%m-%d %H:%M:%S'))
```

isoformat () Return a string representing the date and time in ISO 8601 format, *YYYY-MM-DDTHH:mm:ssZ*. The time zone is always UTC and has a suffix of *Z*.

Date data (Date)

A date value represents date data.

Date type values have the following read-only properties:

Key	Meaning
year	Year
month	Month (values from 1 to 12)
day	Day (value from 1 to the number of days in the given month and year)
weekday	With Monday as 0, Sunday as 6, a value representing the day of the week as an integer

Date data has the following methods:

format (dt_fmt) Converts the date data to a character string in the format specified by dt_fmt. Format specification of this format conforms to C language strftime() function.

Time data (Time)

A time value represents time data.

Time type values have the following read-only properties:

Key	Meaning
hour	Time (value from 0 to 23)
minute	Minute (value from 0 to 59)
second	Seconds (values from 0 to 59)

Time data has the following methods:

format (dt_fmt) Converts the time data to a character string in the format specified by dt_fmt. Format specification of this format conforms to C language strftime() function.

Elapsed time type (Timedelta)

The value of elapsed time type data represents the difference between date and time type values. Addition and subtraction are possible between the date-time type value and the elapsed time type value. Also, the difference between date and time type values will be elapsed time type.

The elapsed time value has the following read-only properties:

Key	Meaning
days	Days
seconds	Seconds
microseconds	Microseconds
total_seconds	The total number of seconds contained in the duration.

4.4 Variables

A variable is a name given to a storage area that holds a value. Variables in the Kompira job flow language can hold values of any type.

Note: Variables are not shared between child processes generated by fork and pfor blocks, or between parent processes and child processes, even if they have the same scope. However, it is possible to reference (read) the variable of the scope of the parent process from the child process when the child process is generated.

4.4.1 Local variables

Local variables are introduced by job flow parameters, assignment jobs. Local variables have different scopes depending on the position on the source code where the variables are introduced.

Job Flow Scope

A job flow scope is a scope that can be referred to from any subsequent job following the job in which the variable is introduced. The job flow parameter has a job flow scope. Also, if an undefined variable is newly introduced by an assignment job, that variable has a job flow scope.

Job flow scope variables are hidden if variables of the same name are redefined in inner block scope.

Block scope

Block scope is a scope that can only be referenced from within that block. Variables defined by simple blocks and loop variables introduced by for and pfor blocks have block scope.

4.4.2 Environment variable

Deprecated since version 1.6: Environment variables have been deprecated in version 1.6.0. Use the $ENV state variable instead.

4.4.3 Special variable

A special variable is a variable with special meaning defined in advance by the system.

Status variable

A status variable is a reserved variable for temporarily storing of execution results such as remote jobs, status codes, etc., and starts with $. These variables are those whose values are set automatically by the Kompira engine and can not be explicitly assigned in the job flow.

There are the following types of status variables:

Variable name	Meaning
$RESULT	Job execution result (standard output)
$STATUS	Job execution status
$ERROR	Job execution error message (standard error output)
$DEBUG	Debug information
$ENV	Environment variable dictionary

Note: The character code of the execution result of the job is automatically determined and converted into an appropriate character string. If conversion to a character string fails, job execution is regarded as failed and an error is returned.

The $ENV state variable contains a dictionary of environment variable fields in an environment variable type (Environment) object set by the JobFlow user.

Control variable

The control variable is a variable for specifying the host name, login name, etc. when executing the remote job, and it is a variable in the form of two consecutive underscores (_) appended before and after, such as __*__ . It is a variable.

The control variable can be defined as a local variable or it can be set as an environment variable.

There are the following types of control variables:

Variable name	Meaning
__realm__	Specify the management area to execute the remote command
__host__	Specify the execution host name of the remote command
__connection__	Specify the host's connection type of remote command execution
__user__	Specify the execution user name of the remote command
__password__	Specify a password
__node__	Specify the node information object to execute the remote command
__account__	Specify the account information object necessary for executing the remote command
__sudo__	When executing in sudo mode, set it to true (the PTY mode is true in this circumstance)
__dir__	Specify the execution directory of the remote command
__port__	Specify ssh port number
__keyfile__	Specify ssh key file path
__passphrase__	Specify the passphrase of the ssh key file (it can be omitted if there is no passphrase or same as __password__)
__timeout__	Specify the number of seconds before the remote command times out
__proxy__	Specify the proxy host when connecting to the execution host
__shell__	Specify the shell to use when executing the remote command (default: "/bin/bash -l -c")
__use_shell__	Set to false if shell is not used when executing remote command
__use_pty__	Set to true to use PTY when executing remote command
__winrs_auth_type__	Specify the authentication method of WinRS connection from "ntlm" (default) and "credssp".
__winrs_scheme__	Specify the scheme of WinRS connection from "http" (default) and "https".
__winrs_use_tls1_0__	Set to true to use TLS 1.0 when performing CredSSP authentication with a WinRS connection. (For environments where TLS 1.2 can not be used, such as Windows Server 2008)

Deprecated since version 1.4: The control variable __via__ has been removed in version 1.4.0. Use __proxy__ instead.

Note: If the directory string specified by __dir__ contains shell metacharacters such as (and) and "and" etc., it must be properly escaped as follows:

```
[__dir__ = 'somedir\\(foo\\)']
```

Note: __dir__ can't be specified when __sudo__=true and __use_shell__=false.

Note: If you do not specify __timeout__ or set a value of 0, timeout does not occur when executing remote command. The operation when __timeout__ is set to a negative value is undefined.

Changed in version 1.4.9.

In winrs mode, as long as the remote command being executed continues to output, it does not time out. In other words, it will time out if there is no output for the number of seconds specified by __timeout__.

Changed in version 1.5.4.post5.

Even when there is command output in winrs mode, it now timeouts in seconds specified by __timeout__.

Note: Changed in version 1.4.8.post6.

For remote command execution in winrs mode, the smaller value of the value specified by __timeout__ and the value set by MaxTimeoutms from WinRM is applied.

Changed in version 1.5.4.post5.

In command execution in winrs mode, the timeout specification by __timeout__ is now prioritized over MaxTimeoutms.

4.5 Expression

Expressions in the job flow program are evaluated during the execution of the job flow and have some value as a result.

4.5.1 Atomic formula

An atomic expression is the basic unit that constitutes an expression. Identifiers, object paths, literals are included in atomic expressions. Also, the format enclosed in parentheses is also grammatically classified as an atomic expression.

```
atomic_expression  ::=   IDENTIFIER | OBJECT_PATH | SPECIAL_IDENTIFIER
                       | literal
                       | parenth_form
                       | array_expression
                       | dict_expression
```

Identifier (IDENTIFIER)

An identifier as an atomic expression represents a variable name. When evaluating the variable name, it returns the value bound to that variable name under the execution environment at the time of evaluation.

Object Path

The object path returns the value of the Kompira object pointed to by that path.

If the object does not exist, a run-time error occurs.

Special identifiers

The special identifier represents a status variable after job execution. At the start of the job flow, $STATUS is initialized to 0 and $RESULT and $ERROR are initialized to the empty string (“”), respectively.

Literal

Literals are strings, binary, integers, floating-point numbers, booleans, and nulls.

```
literal  ::=  STRING | BINARY | INTEGER | FLOAT | BOOLEAN | NULL
```

When evaluating a literal, it becomes the value indicated by that literal.

In the case of a string literal, the variable prefixed by $ in that string is expanded. The following rules will be observed:

- A $identifier is a replacement placeholder specification and corresponds to mapping to the key “identifier”. By default, the “identifier” part must contain Kompira’s identifier. If a character that can not be used as an identifier appears after $, specification of the placeholder name ends.
- ${identifier} is the same as $identifier. It is a necessary writing method if the placeholder name is followed by a character string that can be used as an identifier and you do not want to treat it as part of the placeholder name.

For example, if you execute the following job flow, “Hello Kompira” will be output to the console.

```
[name = 'Kompira']
-> print('Hello $name')
```

Parentheses format

The parenthesis format evaluates the enclosed expression and returns its value.

```
parenth_form  ::=  "(" expression ")"
```

Array expression

Array expressions are a comma-separated list of expressions enclosed in square brackets. It is also possible to omit the sequence of expressions.

```
array_expression  ::=   "[" expression_list? "]"
expression_list   ::=   expression ("," expression_list)*
```

When evaluating an array expression, it returns the data of the newly created array type as a value. Each element of the array is evaluated from left to right.

Dictionary expression

A dictionary expression is a comma-separated sequence of key-value pairs enclosed in curly brackets. It is possible to omit the list of pairs. If the pair is connected with an equal sign, the key must be an identifier. If the key is duplicated, it will report an error at the time of compilation. If it is bound by a colon, the key can describe arbitrary expressions. In this case key duplication is not checked.

```
dict_expression  ::=   "{" ( binding_list | key_val_list )? "}"
binding_list     ::=   binding ("," binding_list)*
binding          ::=   IDENTIFIER "=" expression
key_val_list     ::=   key_val ("," key_val_list)*
key_val          ::=   expression ":" expression
```

When evaluating a dictionary expression, it returns the newly created dictionary type data as a value. Given a set of comma-delimited key-value pairs, the expression will be evaluated from left to right to define the dictionary's entry. Giving a duplicate key, results in a syntax error.

4.5.2 Postfix expressions

Postfix expressions have the highest connectivity among expressions.

```
postfix_expression  ::=   attribute_reference
                        | subscript_reference
                        | function_call
                        | atomic_expression
```

Attribute reference

Attribute references are formats in which the postfix expression has a dot followed by an identifier.

```
attribute_reference  ::=   postfix_expression "." IDENTIFIER
```

The evaluation result of the postfix expression must be an object type. The value specified by the identifier of the object of the post evaluation result is the value of the evaluation result. If the specified attribute does not exist, it becomes the field value of the object with the identifier string as the key. If no such field exists, a runtime error will result.

Subscript Reference

A postfix expression followed by an expression enclosed in square brackets represents an expression that retrieves an element from fields or arrays of objects, and dictionary data.

```
subscript_reference  ::=   postfix_expression "[" expression "]"
```

The evaluation result of the postfix expression must be one of either object type, dictionary type, or array type.

If there is no element corresponding to the key or index, a runtime error will occur.

Function calls

Function calls call functions of objects defined by built-in functions, library type objects, and methods of objects with a list of arguments. The argument list consists of an expression list followed by a binding list (keyword argument list), each of which can be empty.

```
function_call  ::=   postfix_expression "(" argument_list? ")"
argument_list  ::=   expression_list ("," "*" atomic_expression)? ( "," binding_list )?
                     ( "," "**" atomic_expression )?
                     | binding_list ("," "**" atomic_expression)?
                     | "*" atomic_expression ("," binding_list)? ("," "**" atomic_expressio
                     | "**" atomic_expression
```

If the syntax *atomic_expression appears in the function call, atomic_expression must evaluate to an array. Elements from this array are treated as if they were additional positional arguments

If the syntax **atomic_expression appears in the function call, atomic_expression must evaluate to a dictionary, the contents of which are treated as additional keyword arguments.

Each element of the argument list is evaluated before the function call.

4.5.3 Operator expression

Unary operator

Unary operators show as + and -. Since the unary operator is a right join, +-x has the same meaning as +(-x).

```
unary_expression  ::=   postfix_expression
                        | ( "+" | "-" ) unary_expression
```

The unary - operator inverts the sign of the numeric value to be argument.

The unary + operator does not change numeric arguments.

Multiplication and division operator

The multiplicative operator has *, /, and %. All of them have the same priority and become a left join.

```
multiplicative_expression  ::=   unary_expression
                                 | multiplicative_expression "*" unary_expression
                                 | multiplicative_expression "/" unary_expression
                                 | multiplicative_expression "%" unary_expression
```

The * operation is the product of the arguments. If either argument is a character string or an array and one is an integer, it is the value obtained by repeating the number of strings and arrays by that number. For example, the expression 'foo' * 3 evaluates to 'foofoofoo'.

The / operation is the quotient between the arguments. If division by zero occurs, an error occurs.

The % operation is the remainder when dividing the first argument by the second argument when the two arguments are integers. If the first argument is a character string and the second argument is a dictionary, it returns the result of replacing the template string.

Arithmetic operators

Arithmetic operators include + and -. All of them have the same priority and become a left join.

```
additive_expression  ::=   multiplicative_expression
                           | additive_expression "+" multiplicative_expression
                           | additive_expression "-" multiplicative_expression
```

The + operation returns the value obtained by adding the argument. If both arguments are a string or an array, it returns the concatenated value.

The - operation returns the subtracted value between the arguments.

Comparison operators

The comparison operators are <, >, ==, >=, <=, !=, =~ and !~. All of them have the same priority and become a left join.

```
comparison_expression  ::=   additive_expression
                             | comparison_expression "<" additive_expression
                             | comparison_expression ">" additive_expression
                             | comparison_expression "==" additive_expression
                             | comparison_expression ">=" additive_expression
                             | comparison_expression "<=" additive_expression
                             | comparison_expression "!=" additive_expression
                             | comparison_expression "=~" additive_expression
                             | comparison_expression "!~" additive_expression
```

The result of the comparison is Boolean value true or false. You can chain any number of comparisons. For example, x < y <= z is equivalent to x < y and y <= z. However, in this case, y is evaluated only once for the former. Also, x < y <= z and (x < y) <= z have different meanings. The latter compares the Boolean value of z with the result of evaluating x < y.

The meaning of comparison between values of the same type depends on type.

- For integer-by-integer comparisons, an arithmetic comparison is made.
- In comparison between character strings, a dictionary comparison is performed.
- In comparison between arrays, a dictionary comparison is performed using the comparison result of each corresponding element.
- Comparison between dictionaries is defined only for equivalence judgment. They are only equivalent when the keys are in the same order and the corresponding elements of the key and value are equal.

x != y is equivalent to not (x == y).

=~ makes similar comparisons. The meaning depends on type.

- In comparisons between pattern and character strings, comparison by pattern matching is performed.
- In comparison between character strings and other types, comparison is performed by converting values other than character strings into character strings.
- In comparing arrays, similarities between corresponding elements are compared.
- A comparison between dictionaries ignores differences in the order of keys and compares the values corresponding to each key in a similar manner.
- Except for the above, it has the same result as the equivalent comparison by normal ==.

x !~ y is equivalent to not (x ~= y).

Inclusion operators

Inclusion operators are expressed as in or not in. They all have the same priority and become a left join.

```
membership_expression  ::=   comparison_expresion
                           | membership_expression "in" membership_expression
                           | membership_expression "not" "in" membership_expression
```

Inclusive operations x in y returns true if the value x is included as an element of y, and false if it is not included. x not in y is the same as not (x in y).

If y is a value other than an array type, the judgment as to whether it is an element or not is as follows:

- If x and y are both strings, they are regarded as elements if x is a substring of y.
- If y is a dictionary type value, it is considered an element if x is included in the key set of y.
- If y is a directory/table type object, it is regarded as an element if x is a child object of y.

Logical operators

Logical operators include not, and, and or. When a boolean value is required as a result of a logical operation context or expression, false, null, 0, an empty string (“”), an empty array ([]), and an empty dictionary ({}) are all interpreted as false. Any other value is interpreted as true.

```
logical_not_expression  ::=   membership_expression
                            | "not" membership_expression
logical_and_expression  ::=   logical_not_expression
```

```
                                | logical_and_expression "and" logical_not_expression
logical_or_expression   ::=   logical_and_expression
                                | logical_or_expression "or" logical_and_expression
expression              ::=   logical_or_expression
```

The operator “not” is true if the argument is false, if the argument is true, then “not” is false.

The expressions x and y evaluate the expressions x and y respectively, and return the evaluation result of x if x is false. Otherwise it returns the evaluation result of y.

The expression x or y evaluates the expression x and y respectively, and returns the evaluation result of x if x is true. Otherwise it returns the evaluation result of y.

4.6 A job

A job instructs execution of a command, waiting for an event, or a control such as repetition or conditional branching. The syntax of a job is as follows:

```
job  ::=   skip_job
         | execution_job
         | assignment_job
         | update_job
         | event_job
         | builtin_job
         | control_job
         | block_job
```

4.6.1 Skip Job

Skip job does nothing but set $STATUS to 0.

```
skip_job  ::=   "[" "]"
```

4.6.2 Execution job

An execution job performs different processing depending on the value type of the result of evaluating the expression.

```
execution_job  ::=   "[" expression ("<<" expression)? (":" argument_list)? "]"
```

If the result of evaluating the first expression is a character string, the execution job interprets the character string as a command on the remote server or on the local server according to the control context and executes it. If there is a second expression following the symbol <<, the evaluation result of that expression is regarded as a character string and passed to the standard input of the command.

If the result of evaluating the first expression is a job flow object, call that job flow object. If there is an argument list, the value obtained by evaluating the expression of the argument in the list is the parameter of the job flow.

If the result of evaluating the first expression is a script object, that script will be executed on the remote server or on the local server. If there is a second expression following the symbol <<, the evaluation result of that expression is

regarded as a character string and passed to the standard input at the time of script execution. If there is an argument list, the value evaluating the expression of the argument in the list is the command line argument of the script.

If the result of evaluating the first expression is a method of Kompira objects, call its method with argument list as a parameter.

If the result of evaluating the first expression is a library object function, call that function with the argument list as a parameter.

Warning: The length of the command string, the size of the script and script command line arguments is limited to 112 KB. If this limit is exceeded, the job execution will fail and set $STATUS to -1.

4.6.3 Assignment jobs

Assignment jobs assign the evaluation result of the right side of the = expression to a variable.

```
assignment_job  ::=   "[" binding_list "]"
```

If the variable is undefined, a variable with job flow scope is newly defined and initialized with the evaluated value.

4.6.4 Update jobs

An update job evaluates the first expression and updates the contents of variables, objects, and fields as a result of evaluating the target expression against the value of the result.

```
update_job         ::=   "[" expression ">>" target_expression "]"
target_expression  ::=   IDENTIFIER | OBJECT_PATH
                         | target_expression "." IDENTIFIER
                         | target_expression "[" expression "]"
```

4.6.5 Event jobs

When the result of evaluating the first expression is a channel object or a task object, the event job waits for the event of that object. Received objects are stored in $RESULT.

If you pass a process object, wait until the process ends. In this case, $RESULT stores process objects.

If you pass a list whose elements are channels, tasks, or process objects, wait for the event of one of the objects. $RESULT stores a list with two elements described below. The first element of the list is the object where the event occurred, the second element is the received object.

```
event_job  ::=   "<" expression (("?" | "??") expression)? (":" argument_list)? ">"
                 | "<" ">"
```

If ? is followed by an expression (guard expression), it becomes an event job with a guard.

If there is an argument list, timeout is specified. If the value of the first expression is a datetime type, the date and time to time out is specified, and in the case of an integer type, the number of seconds until the timeout is specified. If it times out, $STATUS is set to 1.

If it is empty, it will always fire, so the job flow will continue executing immediately.

4.6.6 Built-in jobs

Embedded jobs are called Kompira's built-in jobs.

```
builtin_job  ::=   IDENTIFIER "(" argument_list? ")"
```

If there is an argument list, the expression is evaluated from the beginning in order of the list, and the result is passed as a parameter of the built-in job.

For a list and details of the embedded jobs provided by Kompira, see *Kompira Standard Library*.

4.6.7 Control jobs

There are two control jobs, break and continue.

```
control_job  ::=   "continue" | "break"
```

Control jobs can only be used inside while blocks and for blocks. If you use it elsewhere, it will result in a compile-time error.

Continue

Continue transfers control to the beginning of the next iteration of the while/for block.

Break

Break aborts the iteration of the while/for block, and transfers control to the block after it.

4.6.8 Block jobs

A block job creates a new block scope.

```
block_job  ::=   simple_block
               | if_block
               | for_block
               | while_block
               | case_block
               | choice_block
               | fork_block
               | pfor_block
               | session_block
               | try_block
```

Simple block

If a simple block has a variable declaration, the local variable holding the block scope is newly defined, and then a job flow expression in the block is executed accordingly. If the variable declaration is omitted, simply execute the job flow expression in the block.

```
simple_block  ::=   "{" (binding_list "|")? jobflow_expression "}"
```

if Block

The if block evaluates the first conditional expression and branches processing depending on the result. If the conditional expression is omitted, the value of $RESULT which is the execution result of the immediately preceding job is used.

```
if_block     ::=   "{" "if" expression? "|" jobflow_expression "}"
                   | "{" "if" expression? "|" then_clause elif_clause* else_clause? "}"
then_clause  ::=   "then" ":" jobflow_expression
elif_clause  ::=   "elif" expression ":" jobflow_expression
else_clause  ::=   "else" ":" jobflow_expression
```

In the first form of an if block, the job flow expression in the block is executed only if the conditional expression is true.

In the second form of an if block, if the first conditional expression is true, the jobflow expression in the then clause is executed. In the case of false, the conditional expression of the next elif clause is evaluated, and if the value is true, the job flow expression of the elif clause is executed. When every conditional expression is false, and if there is a last else clause, the job flow expression of the else clause is executed.

Case block

The case block evaluates the first expression and attempts to match that value with the value evaluated for the pattern expression of each case clause. If the matching is successful, execute the job flow of the corresponding case clause. If multiple pattern expressions of case clause are described in comma-separated form, if matching with any one pattern is regarded as a success, the job flow of that section is executed.

If the first expression is omitted, the value of $RESULT (which is the execution result of the previous job) is used.

```
case_block   ::=   "{" "case" expression? "|" case_clause+ else_clause? "}"
case_clause  ::=   expression_list ":" jobflow_expression
```

The case clause is followed by a pattern expression followed by a colon (:), which is a delimiter, and a job flow expression to be executed when it matches the pattern. If the result of evaluating the pattern expression is a pattern object, matching based on that pattern object is attempted. If the evaluation result of the pattern expression is a character string, it is treated as a case-sensitive Glob pattern. Otherwise, we will do a simple == comparison by matching.

Patterns are tried in order from the beginning of the case clause. If no pattern matches, if there is an else clause, the job flow expression is executed. If there is no else clause, matching is considered to have failed and $STATUS is set to 1.

for block

A for block is used to iterate over elements within an object that contains multiple elements, such as lists, directories, and tables.

```
for_block  ::=   "{" "for" IDENTIFIER "in" expression "|" jobflow_expression "}"
```

Expressions are evaluated only the first time when the for block is executed. The evaluation result of the expression must be a repeatable object or an integer value, otherwise an execution error will occur. Each element of the object is assigned to a local variable indicated by an identifier (IDENTIFIER). If the evaluation result of the expression is an integer value N, the local variable iterates in the range 0 to N-1. However, it does not iterate if N is 0 or negative. Since this local variable has the scope of the for block, it can not be referenced after leaving the for block.

When the break job is executed in the job flow expression, the loop is terminated. When the continue job is executed, the subsequent processing of the job flow expression is skipped and the loop is terminated.

$STATUS at the end of the for block is always set to 0.

While block

The while block evaluates the expression iteratively, and if it is true, it executes the job flow expression. If the expression is false, the while block ends the iteration.

```
while_block  ::=   "{" "while" expression "|" jobflow_expression "}"
```

When the break job is executed in the job flow expression, the loop is terminated. When the continue job is executed, skip the subsequent processing of the job flow expression and return to evaluating the expression.

$STATUS at the end of the while block is always set to 0.

Choice block

The choice block waits for multiple event jobs, and when one becomes executable, it executes the job flow expression following that event job.

```
choice_block          ::=   "{" "choice" "|" eventflow_expression+ "}"
eventflow_expression  ::=   event_job ("->" | "=>" | "->>" | "=>>") jobflow_expression
```

If multiple event jobs can be executed at the same time, the event job closest to the top takes precedence.

Fork block

The fork block starts executing the job flow expression as a child process.

```
fork_block  ::=   "{" "fork" "|" jobflow_expression+ "}"
```

The fork block waits until all child processes that has not been detach() have completed execution. $RESULT is set to the list of child processes. At the end of the fork block, $RESULT is set to the list of all child processes created in the fork block, and $STATUS is set to the number of child processes terminate abnormally. If all child processes

terminate normally, $STATUS is set to 0.

If the process generated by the fork block exceeds the limit of the number of processes, the fork block waits for execution until the other process complete execution and it falls within the process limit.

Pfor block

The pfor block creates a child process and performs concurrent processing on elements in an object including multiple elements such as lists, directories, and tables.

```
pfor_block  ::=   "{" "pfor" IDENTIFIER "in" expression "|" jobflow_expression "}"
```

Expressions are evaluated only during the first time when executing the pfor block. The evaluation result of the expression must be a repeatable object, otherwise it will result in an execution error. A child process is created for each element of the object and the corresponding element in each child process is assigned to a local variable identified by an identifier (IDENTIFIER), and execution of the child process is started. If the evaluation result of the expression is an integer value N, execution of the child process is started for each of the local variables from 0 to N-1. However, if N is 0 or negative, the child process is not executed.

The pfor block waits until all child processes that has not been detach() have completed execution. $RESULT is set to the list of child processes. At the end of the pfor block, $RESULT is set to the list of all child processes created in the pfor block, and $STATUS is set to the number of child processes terminate abnormally. If all child processes terminate normally, $STATUS is set to 0.

If the process generated by the pfor block exceeds the limit of the number of processes, the pfor block waits for execution until the other processes are executed and it is within the processing limit.

Session block

The session block starts a session with the remote server.

```
session_block  ::=   "{" "session" IDENTIFIER "|" jobflow_expression "}"
```

When a session block is executed, it first starts a session with the remote server specified by the control variable. The session channel for interaction with the remote server in the session is assigned to the local variable indicated by the identifier (IDENTIFIER). When sending (send) a character string to this session channel, a character string is sent to the remote server side. In addition, in order to obtain output from the remote server side, data is acquired from the session channel using the event job. The output from the remote server is stored in the session channel as a line-by-line message. Therefore, reading messages from the session channel is one line at a time.

Exiting the session block ends the session, and the session channel is closed. After that, transmission to the session channel will result in an error. Reading messages from the session channel also results in an error. (However, messages output from the remote server before closing the session can be read)

Calling break in a session block closes the session and ends the block.

When the session block ends normally, $STATUS is set to 0. In addition, the session channel is stored in $RESULT. Unread data is stored in the data attribute of the session channel. (Each line of the message is concatenated and becomes one character string data)

If the session fails to start, the session block is terminated without being executed in the session block, and $STATUS is set to non-0. In addition, $ERROR contains a message indicating the cause of the error.

Note: You can execute a command job within a session block, but you can not start another session anew.

Note: For session blocks, currently only connections in ssh mode are supported. Execution of the session block after setting the control variable in the winrs mode or the local mode causes an error.

The following shows an example of a job flow program that executes interactive processing by executing the su command.

```
[__host__ = 'server.exmaple.com', __user__ = 'testuser', __password__ = 'password',
 __use_pty__ = true   # su command require PTY.
] ->
# log in to server.example.com and start session.
{ session s |
    [s.send: 'LANG=C su\n'] ->              # execute su command
    <s ?? 'Password: '> ->                  # wait for password prompt
    [s.send: 'root_password\n'] ->          # send root's password
    <s ?? g'*]# '> ->                       # wait for root user prompt
    [s.send: 'service httpd restart\n'] ->  # restart the httpd service
    <s ?? g'*]# '> ->                       # wait for root user prompt
    [s.send: 'exit\n']                      # exit from root
} ->
print('OK')
```

Try block

The try block catches the abnormal termination that occurred while executing the job flow in the block, and continues the processing.

```
try_block  ::=   "{" "try" "|" jobflow_expression "}"
```

If the job flow enclosed by the try block ends normally, the try block sets $STATUS to 0, and if it ends abnormally, sets $STATUS to 1. It also stores debugging information in $DEBUG.

If exit is called while executing the job flow in the try block, the job flow always ends. Also, if the execution of the job flow is cancelled by the user while executing the job flow in the try block, the job flow will terminate execution.

4.7 Job flow expressions

A job flow expression is an expression that combines jobs with connectors.

```
jobflow_expression  ::=   jobflow_expression "->" job
                        | jobflow_expression "=>" job
                        | jobflow_expression "->>" job
                        | jobflow_expression "=>>" job
```

4.7.1 Connectors

There are multiple types of connectors, and whether job flow processing continues or not when the job fails is different.

Below is a list of connectors, the behaviour they exhibit when the job fails and the value of the status variable, when processing is continued.

Connectors	Command abnormal termination	Remote login failed
->	Forced termination	Forced termination
=>	Processing continuation $STATUS = 1-255 $RESULT = (stdout) $ERROR = (empty)	Forced termination
->>	Forced termination	Processing continuation $STATUS = -1 $RESULT = (empty) $ERROR = (error message)
=>>	Processing continuation $STATUS = 1-255 $RESULT = (stdout) $ERROR = (empty)	Processing continuation $STATUS = -1 $RESULT = (empty) $ERROR = (error message)

If the execution status of the remote command is anything other than 0, it begins to operate as per "Command abnormal termination" in the above table. At this time, the value of the execution status of the remote command is the value of $STATUS.

If ssh times out, or if the IP address, user name, password, etc. specified in the job flow are incorrect, the failure will be as per "Remote login failure" in the above table.

Changed in version 1.5.4.post5: When remote login fails, $ERROR contains a message indicating the cause of the error.

4.8 Job flow Program

A job flow program consists of zero or more parameter declarations followed by job flow expressions. If the job flow expression is empty, execution of the job flow program can be skipped.

```
jobflow_program  ::=   (parameter_declaration)* jobflow_expression?
```

4.8.1 Parameter declaration

The parameter declaration takes the following form:

```
parameter_declaration  ::=   "|" IDENTIFIER ("=" expression)? "|"
```

In the parameter declaration, if there is a form of parameter_declaration = expresssion, the job flow has default parameters. For parameters with default values, if the corresponding parameter is omitted during the job flow call, the value of the parameter will be replaced with the default value. The default parameter expression is evaluated for each job flow invocation.

CHAPTER

FIVE

KOMPIRA STANDARD LIBRARY

Author Kompira development team

In this library reference manual, the Kompira Standard library will be explained.

5.1 Built-in functions / jobs

Kompira's jobs are predefined as built-in jobs and built-in functions.

Embedded jobs are divided into two types: local embedded jobs not run via the job manager and remote embedded jobs executed by the job manager.

5.1.1 Local embedded jobs

Local embedded jobs are executable jobs even if the job manager is not running.

self () Re-executes its own job flow from the beginning. When re-executing, the parameters of the job flow are not changed. Also, in the case of a job flow whose multiplicity is specified, re-executes with the lock held.

print ([*message*, [*args*, ...]]) Outputs a message string to the console and carries out a line feed.

When multiple arguments are given, multiple message strings are separated by space characters and output. If you omit all arguments, only newlines are used.

sleep (*timeout*) Sleeps the process for the number of seconds specified by the timeout. If the timeout is a datetime type, it will sleep until that date and time.

exit ([*status=0*, [*result=''*, [*error=''*]]]) Finishes the process. You can also specify an exit status code with status. Specify the execution result at process termination with result. You can also specify an error message with error.

return ([*result=''*, [*status=0*, [*error=''*]]]) Returns control to the caller of the job flow. Specifies the execution result with result. You can also specify an exit status code with status. You can also specify an error message with error.

abort ([*message*]) It outputs a message to the console and abnormally ends the job. The end status code is set to 1.

assert (*value*, [*message*]) It verifies that value is true, otherwise it outputs a message to the console and abnormally ends.

detach () Separates the running process of the child process from the parent process. This allows the parent process to proceed further without waiting for the child process to finish.

suspend () Pauses a running process.

urlopen (*url*, [*user*, *password*, *data*, *params*, *files*, *timeout*, *encode*, *http_method*, *verify*, *quiet*, *headers*, *cookies*, *charset*, *binary*, *prox*

It sends a HTTP request to the url specified by the argument and gets the result.

When user and password are specified, access by basic authentication is performed.

For data, you can specify the data to send with a POST request as a dictionary type. Transmission data is encoded in the method specified by the encode argument.

Passing a dictionary to params expands it as a URL query string. For example, if you call as follows, the URL actually accessed is `http://example.com?key1=value1&key2=value2`.

```
urlopen(url='http://example.com', params={key1='value1', key2='value2'})
```

You can pass files to upload to files. The file can be specified in either a dictionary with the fields name and data, a list of file names and contents, a filename on the Kompira server, or an attachment field.

```
files={file={name='filename', data='content'}}
files={file=['filename', 'content']}
files={file="/tmp/filename.xls"}
files={file=./attached_file.attached1}
```

In this case, specify the dictionary key ("file" in the above) to be sent to "files" according to the name of the file field of the destination form. For file fields that accept multiple files, you can also pass field names and files side-by-side in list format.

```
files=[['file', {name='filename1', data='content1'}],
       ['file', {name='filename2', data='content2'}]]
```

In this case, please also specify the inner field name and file in the list. When files are specified, they are encoded in multipart/form-data format.

For timeout, specify the time until timeout in seconds.

For encode, "json" can be specified as the encoding type. When the data specified and encoded is in "json", application/json is automatically set in the Content-Type: header of the HTTP request. If the encode argument is omitted, the transmitted data is encoded in application application/x-www-form-urlencoded format. If files are specified, specifying "json" for encoding will result in an error.

For http_method, specify the method of HTTP request from 'GET', 'POST', 'PUT', 'DELETE', 'HEAD'. If the http_method is omitted, it is POST method if data or files are specified, and GET method if not specified.

When verify is set to true, a SSL certificate check is performed when the specified URL is https accessed. If an illegal SSL certificate is detected, the urlopen job will generate an error. The default value of verify is false.

If quiet is set to true, when the verify option is true, suppress warning messages displayed when accessing https.

For headers, you can pass the header information set in the HTTP request as a dictionary type value.

In cookies, pass the cookie passed to the server as a dictionary type value.

For charset, you can specify the character code you expect as a response.

If you need to send an HTTP request via a proxy server, you pass the proxy server URL dictionary to the proxies parameter for the following example. :

```
[proxies = {'http': 'http://10.10.1.10:3128', 'https': 'http://10.10.1.10:1080'}]␣
↪->
urlopen('http://www.kompira.jp', proxies=proxies)
```

Whether or not the acquired content is binary is determined by Content-Type. When Content-Type starts with image | audio | video, or when octet | binary is included, it is judged to be binary. However, if binary is set to true, content is treated as binary regardless of the Content-Type.

This built-in job returns a dictionary type value with the following elements:

Field name	Meaning
url	Response URL
code	Resulting status code
version	HTTP version (If HTTP 1.1, it will be 11)
text	Content of the response (The response body was decoded into text based on the encoding information, but in the case of binary content it will be empty)
content	Content of the response (body of the response as it is binary)
body	Content of the response (When it judges that the content is binary, it becomes the same value as content, and when judging it is text it will be the same value as text)
encoding	Encoding information
headers	Header information (dictionary type) included in the response
cookies	The cookie value (dictionary type) passed from the server
history	When there is a redirect, its history information (list type)
binary	True value of whether it is binary content

mailto (*to*, *from*, *subject*, *body*, [*cc*, *bcc*, *reply_to*, *html_content*, *attach_files*, *parents*, *headers*, *charset*, *reply_to_all*, *inline_content*, *j*

Send mail.

For to, specify the destination mail address as a character string. If you want to send to multiple addresses, specify it in the list with the destination mail address as an element. For from, specify the mail address of the sender. For subject, specify the character string of the mail subject. For body, specify the character string of the mail body. For cc and bcc, specify Cc / Bcc destination email addresses respectively. If you want to specify multiple addresses, pass them in a list. For reply_to, specify the reply mail address.

When html_content is specified, mail of HTML format (text / html) is sent. If html_content is omitted or null is specified, mail with body in text format (text / plain) will be sent. If both body and html_content are null, the mailto job will fail. In attach_files, you can pass a list of file objects or file objects to attach to the mail.

If you pass a parent message (dictionary as a result of mail_parse) to parents, a reply mail will be sent for that message. At this time, the mail headers In-Reply-To: and References: are properly set. In addition, the address set in Reply-To: or From: of the parent message (if set) is set as the destination. If you want to refer to more than one parent message, please pass it as a list. When reply_to_all is set to true when replying to mail with parents, it sets "To:" of parent message as handling "Reply to all" and inherits the destination specified by Cc:.

Passing a dictionary to headers, adds each key of the dictionary as a header item to the mail header.

For charset, you can specify the character code of sending mail. default is UTF-8.

If inline_content is set to true, inline expansion of the attached file occurs. At this time, the MIME mixed subtype of the mail body is "related" and the Content-Disposition header of each attachment is "inline". In addition, "%{Content-ID#num}" (the num part is the index of the attachment specified by attache_files) or "%{Content-ID:filename}" in the body of the body (body, html_content) File name of the file) and the specified placeholder, will be replaced with the Content-ID automatically appended to each attachment (with '<' and '>' removed at both ends). For example, if you attach one image file with attach_files and include the description '' in html_content, The image file attached to is now displayed inline.

If as_string is set to true, instead of actually sending the mail, it converts the entire message including the mail header into a string. The resulting string can be referenced with $RESULT.

The User-Agent header of the mail sent by the mailto job is "Kompira ver X.XX". The "X.XX" part contains the version number of Kompira.

Note: If from is omitted, the sender's e-mail address is determined with the next priority.

(1) process owner's e-mail address

(2) administrator e-mail address of /system/config

(3) webmaster@localhost

download (*from_file*, *to_path*) Download the file in the attached file field to the specified path. For from_file, specify the download source attachment field object.

For to_path, specify the file path of the download destination. The download destination is the file system on the server on which the job manager is running. If the downloaded file path points to a directory, the file name is the file name of the attached file.

The following, downloads the file attached to the attached field of the Kompira object /root/Package to the local /tmp directory. :

```
download(from_file=/root/Package.attached, to_path='/tmp/')
```

upload (*from_path*, *to_object*, *to_field*) Upload the file specified in the attached file field. The result returns the file information of the attached file. For from_path, specify the file path of the download source. For to_object, specify the Kompira object to which you want to attach, and specify the attachment field name of the attached Kompira object with to_field.

The following uploads the locally placed file /tmp/foo.tar.gz to the attached field of the /root/Package object. :

```
upload(from_path='/tmp/foo.tar.gz', to_object=/root/Package, to_field='attached')
```

5.1.2 Remotely embedded job

Remote embedded jobs are built-in jobs that run through the Job Manager. If the job manager is not running, execution waits until the job manager is started.

In remote embedded jobs, the connection information of the remote host is referenced from the control variable.

put (*local_path*, *remote_path*)

Transfer the file from the host on which the job manager is running to the remote host. The result will return a list of destination file paths.

For local_path, specify the source file path. It is also possible to transfer multiple files using wildcards. If local_path is specified as a relative path, it is relative to the directory in which the job manager is running (usually the root directory).

For remote_path, specify the directory path or file path of the transfer destination. If remote_path is specified as a relative path, it is relative to the login user's home directory or relative to the path specified by the __dir__ control variable.

get (*remote_path*, *local_path*)

Transfer the file from the remote host to the host on which the job manager is running. The result will return a list of destination file paths.

For remote_path, specify the source file path. It is also possible to transfer multiple files using wildcards.

For local_path, specify the file path or directory path of the transfer destination (job manager side).

reboot ([*wait=120*]) Restarts the remote host.

In wait, specify the maximum time (in seconds) to wait for the remote host to restart.

The reboot job can only be run by users who run sudo jobs.

5.1.3 Built-in functions

Built-in functions are functions that can be used as Kompira expressions.

In addition to describing it in an expression, it can also be used alone as with embedded jobs. When used alone, the result is inserted in $RESULT.

now () Get the current time.

current () Retrieves its own process object currently executing.

channel () Create an on-memory channel object for sending and receiving data between multiple processes.

datetime (*dt_str_or_date*, [*dt_fmt_or_time*, *zone*]) It converts the character string specified by dt_str_or_date into date and time data. It is also possible to specify a format string with dt_fmt_or_time. By passing date type data to dt_str_or_date and passing time type data to dt_fmt_or_time, you can configure date and time type data combined. The option parameter zone specifies a time zone.

The format conforms to C language strftime() function. An example is shown below:

```
[dt = datetime('2015-1-1 10:30:05', '%Y-%m-%d %H:%M:%S', 'Asia/Tokyo')] ->
print(dt) ->
[dt2 = datetime(dt.date, dt.time)] ->
print(dt2)
```

Note: If dt_fmt is omitted, the format of the date string will be converted as ISO 8601 format as shown below.

YYYY-MM-DD[T]hh:mm(:ss(.mmmmmm)?)?([Z]|[+-]hh(:)?mm)?

You can use T or blank separator for date and time. Specifying seconds, microseconds, and time zones is optional.

If zone is omitted, the local time zone is assumed to be specified.

---

date (*date_str*, [*dt_fmt*]) It converts the character string specified by date_str into date data. It is also possible to specify a format string with dt_fmt.

The format conforms to C language strftime() function.

time (*time_str*, [*dt_fmt*]) It converts the character string specified by time_str into time data. It is also possible to specify a format string with dt_fmt.

The format conforms to C language strftime() function.

timedelta (*days=0*, *hours=0*, *minutes=0*, *seconds=0*, *microseconds=0*) Creates data showing elapsed time. Values of timedelta type and datetime type can be added or subtracted.

int (*x=0*) Converts the string given by argument x to integer type.

float (*x=0.0*) Return a floating point number constructed from a number or string x.

pattern (*pattern*, *typ='r'*, *mode=''*) Creates a pattern object given by the argument pattern. typ represents the type of pattern, you can specify either 'r' (regular expression pattern), 'g' (glob pattern), 'e' (exact match pattern). If 'i' is specified for mode, pattern matching is not case sensitive.

path (*str_or_obj*, [*args*, . . .]) Returns the actual Kompira object from the character string str_or_obj representing the path name. str_or_obj can be an array of strings. When an array or multiple arguments are given, each element is combined and interpreted as a path name.

The following example enumerates the objects directly under the root directory. An example of use:

```
{ for p in path('/') | print(p) }
```

If a relative path is specified for str_or_obj, it refers to the Kompira object relative to the directory where this job flow resides. The following example displays the path of the directory where this job flow resides.

Example usage:

```
print(path('.'))
```

You can also specify a Kompira object for str_or_obj. The following example enumerates Kompira objects at the same level as the object contained in the parent directory of the Kompira object specified by parameter 'dir', that is, the object specified by 'dir'.

Example usage:

```
|dir = /home/guest|
{ for sibling in path(dir, '..') | print(sibling) }
```

user (*user*) Returns a User object with the user name user. Giving an integer value to user returns a User object with that value as user ID. Giving User object returns it as is.

group (*group*) Returns a Group object with the group name, group. Giving an integer value to group returns a Group object with that value as the group ID. Giving a Group object will return it as is.

string (*obj*) Converts object obj to a string.

bytes ([*b*, [*encoding='utf-8'*]]) If b is an integer value, this generates a binary (sequence of bytes) of length b with each byte value of 0. If b is a string, produces a binary encoded with the encoding specified in encoding. If b is an array of integers from 0-255, it generates the corresponding binary. If b is binary, return its value as is.

type (*obj*) Returns the type name of the object obj.

decode (*data*, [*encoding='utf-8'*]) Decodes the binary data, data into a character string with the character code system specified by encoding.

encode (*message*, [*encoding='utf-8'*]) Encodes the string message into binary data with the encoding system specified by encoding.

length (*obj*) Gets the length of the array passed in obj.

has_key (*obj*, *key*) Checks whether dictionary data and objects passed by obj are field accessible with the specified key, key.

json_parse (*data*) Converts a string serialized in JSON format into an object of Kompira.

Example usage:

```
[str = '[1,2,3,true,"foo","bar"]']
-> [obj = json_parse(str)]
-> { for elem in obj | print(elem) }
```

json_dump (*obj*) Convert Kompira's object to a serialized string in JSON format.

mail_parse (*data*) Converts MIME formatted string to Kompira's dictionary object.

In addition to the header information of the mail, you can access the file name with the 'Filename' key in the body of the mail with the 'Body' key. (If the attached file does not exist, 'Filename' key is null)

The body of the mail is encoded in utf-8 format only when Content-Type is text/plain and it is not an attached file.

If the Content-Type is multipart, the 'Is-Multipart' key becomes true and the element of the 'Body' key becomes an array of Kompira dictionary objects.

iprange (*address*) Converts CIDR notation network address to IP network object.

Example usage:

```
{ for ip in iprange('192.168.0.1/24') |
    [__host__ = ip ] ->
    ['hostname'] ->> []
}
```

> **Warning:** The embedded job iprange() will be removed in the near future.

5.2 Kompira objects

Various data handled by Kompira is stored as a Kompira object on the Kompira file system with directory structure. Kompira objects have unique fields and methods for each type, and can be operated from the job flow.

5.2.1 Field type

The types that can be used in fields of Kompira object are as follows. The right side of the colon (:) indicates the type of data when referring to the field from the job flow.

String [String] Represents a field of a string.

Binary [Binary] Represents a field of the binary, entered in hexadecimal notation.

Integer [Integer] Represents an integer field. Values other than integers can not be entered. If it is not entered, it will be null value.

Float [Float] Represents a floating point number field. If an integer is entered, it is converted to a floating point number. If not entered, a null value is returned.

Boolean [Boolean] Represents a boolean field. It is displayed as a check box on the form, corresponding to true when checked and false when unchecked.

Enum [String] Represents a choice field. The list of choices is specified by the field qualifier.

Text [String] Represents a text field.

LargeText [String] Represents a larger text field.

Password [String] Represents a password field. The string is hidden when displaying the field.

File [File] Represents an attachment field. You can upload and download attached files.

Object [Object] Represents a Kompira object field. You can choose Kompira objects from choices. By specifying the field qualifier, it is also possible to restrict choices to objects of a specific type or to objects under a specific directory.

Datetime [Datetime] Represents a date / time field. The format of the date and time information to enter is as follows.

Format	Example
%Y-%m-%d %H:%M:%S	2006-10-25 14:30:59
%Y-%m-%d %H:%M	2006-10-25 14:30
%Y-%m-%d	2006-10-25
%m/%d/%Y %H:%M:%S	10/25/2006 14:30:59
%m/%d/%Y %H:%M	10/25/2006 14:30
%m/%d/%Y	10/25/2006
%m/%d/%y %H:%M:%S	10/25/06 14:30:59
%m/%d/%y %H:%M	10/25/06 14:30
%m/%d/%y	10/25/06

Date [Date] Represents a date field.

Time [Time] Represents a time field.

IPAddress [String] Represents an IP address field. It corresponds to input of IPv4 address format.

EMail [String] Represents a mail address field.

URL [String] Represents a URL field.

Array<T> [Array<T>] Represents an array field whose type of elements is T. You can enter multiple elements of type T. (Array is synonymous with Array<Strring> field.)

The only types that can be specified for the type variable T are String, Binary, Integer, Float, Boolean, Enum, Password, Object, Datetime, Date, Time, IPAddress, EMail, and URL.

Dictionary<T> [Dictionary<T>] Represents a dictionary field. Multiple keys and values can be entered. The type of the value is T. (Dictionary is synonymous with Dictionary<String> field.)

New in version 1.6.0: Binary, Float, Array<T>, and Dictionary<T> fields have been added newly.

5.2.2 Field qualifier

Field modifiers add more control and constraints on field display to field types. The field qualifier is described in the form of a JSON object as shown below.

```
{ "<qualifier1>" : <value1>, "<qualifier2>" : <value2>, ... }
```

The following types of field modifiers exist.

default: All fields Specify the default value for the field.

invisible: All fields Hides fields from forms and views.

help_text: All fields Describes the field. If this modifier is specified, the text specified when editing the object is displayed.

object: Object type fields In the object type field, narrows down the choices. When you describe the path of a type object, the object with that type is displayed as a choice. Also, if you specify a directory or table path, child objects of the object are displayed as choices.

The following example shows job flow type objects as options.

```
{ "object" : "/system/types/Jobflow" }
```

If you do not specify this modifier, all objects will be displayed as choices.

directory: Object type fields In the object type field, narrow down the choices. If you describe a directory or table path, its descendant objects are displayed as choices.

If you specify a path with "~" or "~(username)" at the beginning, that part will be expanded to the user's home directory. If the user name is omitted, the user who is logged in becomes the target.

By specifying it together with the qualifier object, it is possible to select an object of a specific type under a certain directory, etc.

E.g.

```
{ "object" : "/system/types/NodeInfo", "directory" : "~" }
```

However, if you specify a directory that does not exist or does not have read permission, it will be ignored.

no_empty: Object type fields Does not allow empty choices on the input form of Object type field.

E.g.

```
{ "object" : "/system/types/TypeObject", "no_empty" : true }
```

enum: Enum type fields In the Enum type field, specifies a list of character strings to be selected.

Example

```
{ "enum" : ["Server", "Switch", "Router"] }
```

If you want to separate the stored data from the display name, you can specify it as follows by using the pair ["<data>", "<display name>"]. :

```
{ "enum" : [["SV", "Server"], ["SW", "Switch"], ["RT", "Router"]] }
```

pattern: String type fields In the String type field, specify a pattern with a regular expression.

min_length, max_length: String type fields In the String type field, specify minimum and/or maximum length.

min_value, max_value: Integer type fields In the Integer and Float type field, specifies minimum and/or maximum value.

file_accept: File type fields In the File type field, specifies a selectable file types.

E.g.

```
{ "file_accept" : ".xls" }
```

When more than one file type is specified, it is specified by a list.

E.g.

```
{ "file_accept" : [".png", ".jpg"] }
```

Note: The Array<T> and Dictionary<T> fields allow you to specify a qualifier for the element T-type.

New in version 1.5.1: New field qualifiers: pattern, min_length, max_length, min_value, max_value, file_accept have been added.

5.2.3 Properties

Kompira objects provide the following properties.

id The value of the ID of the object. The object ID is a unique integer value that is automatically assigned when the object is created. It can not be updated.

abspath The absolute path value of the object. It can not be updated.

name The value of the object name. The format of the character string that can be used for the object name is the same as the identifier in the Kompira job flow language. You can not create objects with the same name in the same directory.

description A character string that describes the object.

display_name The display name of the object. The display name string has no format restriction unlike object names.

field_names A list of field names that the object has. It is a list type value. It can not be updated.

owner The owning user of the object. It is a user object.

created The creation date and time of the object. It will be a date and time type value. It can not be updated.

updated The update date and time of the object. It will be a date and time type value. It can not be updated.

parent_object The parent object of the object, i.e. the directory (or table) object. It can not be updated.

children A child object list of objects. If the object does not have child objects, such as when it is not a table or directory, it is an empty list. It can not be updated.

type_object The type object of the object. It can not be updated.

type_name The type name of the object. It can not be updated.

user_permissions User permission information. It is a dictionary type object with key as writable, readable, executable as the key.

group_permissions Group permission information. It is a dictionary type object with writable, readable, executable and priority keys

5.2.4 Method

Kompira objects provide the following methods.

delete Deletes the object.

update [[*key1*=*val1*, *key2*=*val2*, ...]] Updates the values of the fields key1, key2, ... of the object to val1, val2, ...

rename [*name*] Change the name of the object to name.

5.3 Built-in objects

This section describes standard type objects pre-built in Kompira.

An object of Kompira has a type indicated by a type object. For example, a job flow object has a job flow type and a directory object has a directory type. In Kompira, types such as job flow type and directory type are also defined as objects, so they also have types of type objects. I.e. the type of the type object is a type object.

Kompira's objects have fields and methods specific to that type.

5.3.1 Type Object (TypeObject)

The type object type defines the fields and methods of Kompira objects belonging to that type. By defining a new type object, the user can freely add the type of Kompira object.

Note: When modifying a type object, such as adding a field to an existing type or deleting an unnecessary field, Kompira processes it according to the following rules.

- Fields deleted by the changed type object are ignored and become inaccessible.
- Newly added fields of the new type object are automatically initialized with null values.

Field

In the type object type, the following fields are defined.

extend (extension module) [String] Specifies Python extension module paths referenced by type objects. The default is kompira.extends.

To extend the behavior and view of type objects, create extension model modules models.py and extension view modules views.py as Python modules and place them under the path specified here.

fieldNames (field names column) [Array] Specifies an array of field names of objects of this type as an array. The rules for strings that can be used in field names are the same as the job flow language identifier.

fieldDisplayNames (field display names column) [Array] Specifies the list of field display names of objects of this type as an array. An arbitrary character string can be used for the field display name. The order of array elements must correspond to the columns of field names.

fieldTypes (field types column) [Array] Specifies an array of field types of objects of this type as an array. The order of array elements must correspond to the columns of field names.

Method

Methods specific to type object types are not specifically defined.

5.3.2 Directory (Directory)

Directory type specifies the type of directory object. You can have several different types of Kompira objects under a directory object. This allows Kompira objects to have a hierarchical structure as well as Unix file systems.

Field

A unique field is not defined for the directory type.

Method

The following methods are defined in the directory type.

add [*name*, *type_obj*, [*data*, *overwrite*]] Under the directory, add a type_obj type Kompira object with the name specified by name. Dictionary type data can be passed to data, so that you can initialize the field value of the object. $RESULT stores newly added objects. If you pass true to the overwrite argument, even if an object of the same name exists under the directory, it does not cause an error and updates the object.

move [*obj*, [*name*]] Moves the object specified by obj under the directory. If name is specified, the name of the object to be moved is changed to name.

copy [*obj*, [*name*]] Duplicates the object specified by obj under the directory. If name is specified, the name of the duplicated object is changed to name. If obj is a directory or table, child objects are recursively duplicated. $RESULT stores newly created objects.

has_child [*name*] Returns true if the child object specified by name exists under the directory, false if it does not exist.

find [*args*] If objects matching the condition specified by args exists under the directory, the objects is returned as a list.To filter by attribute of the objects, you can specify filters `` <attribute-name> = <value> `` in args.

For example, if you want to get a list of type objects, specified as shown below.

```
[result = /.find(type_object=/system/types/TypeObject)]
```

Although the above is filtering by exact matching, detailed filtering conditions can be specified by describing the attribute name followed by a lookup as follows.

For example, you can filter objects that contain `kompira` in the path.

```
[result = /.find(abspath__contains='kompira')]
```

The lookup types and filtering method is as follows.

Lookup	Filtering method
`exact`, `iexact`	The attribute exactly matches the specified value.
`contains`, `icontains`	The attribute contains the specified value.
`startswith`, `istartswith`	The attribute starts with the specified value.
`endswith`, `iendswith`	The attribute ends with the specified value.”
`regex`, `iregex`	The attribute matches the specified regular expression.
`gt`, `gte`	The attribute is greater than specified value (`gt`). The attribute is greater than or equal to the value specified (`gte`).
`lt`, `lte`	The attribute is less than specified value (`lt`). The attribute is less than or equal to the value specified (`lte`).
`in`	The attribute is included in the specified values.

In filtering others than virtual objects by attribute value, the lookup that can be specified depends on attribute.

Attribute	Specifiable Lookup
owner	`exact`, `in`
abspath	`exact`, `iexact`, `contains`, `icontains`, `startswith`, `istartswith`, `endswith`, `iendswith`, `regex`, `iregex`
display_name	`exact`, `iexact`, `contains`, `icontains`, `startswith`, `istartswith`, `endswith`, `iendswith`, `regex`, `iregex`
description	`exact`, `iexact`, `contains`, `icontains`, `startswith`, `istartswith`, `endswith`, `iendswith`, `regex`, `iregex`
created	`exact`, `gt`, `gte`, `lt`, `lte`
updated	`exact`, `gt`, `gte`, `lt`, `lte`
type_object	`exact`, `in`

In filtering virtual objects by attribute value, the lookup that can be specified depends on the data type of the attribute.

Type of the attribute	Specifiable Lookup
String	`exact`, `iexact`, `contains`, `icontains`, `startswith`, `istartswith`, `endswith`, `iendswith`, `regex`, `iregex`
Integer	`exact`, `gt`, `gte`, `lt`, `lte`
Datetime	`exact`, `gt`, `gte`, `lt`, `lte`
Object	`exact`
User	`exact`
Boolean	`exact`

If lookup is not specified, `exact` is applied."

If type objects are specified by `type_object` attribute filtering, you can also specify filtering conditions by field value.When filtering by field value, use args as `fields = {<attribute-name> = <value>}` or `fields = {<attribute-name>__<lookup> = <value>}`.

An error occures if you specify filtering by field value when type object is not specified.

For example, if you want to get a list of jobflows that contain `urlopen` in its source code, specified as shown below.

```
[result = /.find(type_object=/system/types/Jobflow, fields={source__contains=
↪'urlopen'})]
```

The available lookups for filtering by field are shown below.

Type of field	Specifiable Lookup
String	`exact`, `iexact`, `contains`, `icontains`, `startswith`, `istartswith`, `endswith`, `iendswith`, `regex`, `iregex`, `in`, `range`
Integer	`exact`, `isnull`, `gt`, `gte`, `lt`, `lte`, `in`, `range`
Boolean	`exact`
Datetime	`exact`, `isnull`, `gt`, `gte`, `lt`, `lte`, `range`
Object	`exact`, `isnull`
File	(same as string)
Array	(same as string)
Dictionary	(same as string)

`exact` is applied when lookup is not specified.

glob [*pattern*] If objects matching the condition specified by patterns exists under the directory, the objects is returned as a list.To filter by patterns of the objects, you can specify as shown below.

```
"<object name>"
```

For example, you can filter objects that contain `kompira` in the path.

```
[result = /.glob("kompira*")]
```

In addition to the object name, it is possible to specify the following elements.

- Path
- Object
- Owner
- Attribute filtering
- Field value filtering

If a path is specified, the object under that path is returned. The pattern is described in the following format.

```
"<path>/<object name>"
```

You can specify /* and /** as the path. Each matches a single-tiered directory and a directory of any depth.

For example, if you want to get a list of objects whose name begin with `kompira` and that contain `user` in path, specified as shown below.

```
[result = /.glob("/**/user/**/kompira*")]
```

If a type object is specified, the object whose type is specified type object is retuened. The pattern is described in the following format.

```
"<object name>.<type object>"
```

For example, if you want to get a list of jobflows, specified as shown below.

```
[result = /.glob("*.Jobflow")]
```

If a owner is specified, the owner's object is returned. The pattern is described in the following format.

```
"<object name>@<owner>"
```

For example, if you want to get a list of root's objects, specified as shown below.

```
[result = /.glob("*@root")]
```

If a attribute filtering is specified, matched object is returned. The pattern is described in the following format.

```
"<object name>(<attirubute name>=<value>)" or "<object name>(<attibute name>_
↪<lookup>=<value>)"
```

Refer to the find method for a list of lookups that can be specified for attribute values.

For example, if you want to get a list object whose display name contains `kompira`, specified as shown below.

```
[result = /.glob("*(diplay_name__contains='kompira')")]
```

If a field value filtering is specified, matched object is returned. The pattern is described in the following format.

```
"<object name>[<field name>=<value>]" or "<object name>[<field name>_<lookup>=
↪<value>]"
```

Refer to the find method for a list of lookups that can be specified for field values.

For example, if you want to get a list of jobflows that contain `urlopen` in its source code, specified as shown below.

```
[result = /.glob("*[source__contains='urlopen']")]
```

These can also be specified in combination. The pattern when all specified is as follows.

```
"<path>/<object name>.<type object>@<owner>(<attirubute filtering>)[<field value␣
↪filtering>]"
```

For example, if you want to get a list of object like below, specified as shown below.

- Objects under /user/app
- It owned by root.
- Its name starts with `Kompira`.
- Jobflow
- <Its display name contains *Kompira*.
- Its multiplicity is 1 or less.

```
[result =
/.glob("/user/app/**/kompira*.Jobflow
@root(display_name__contains='kompira')[multiplicity__lt=1]")]
```

5.3.3 License

License type defines objects that manage Kompira's license file.

Field

A unique field is not defined for the license type.

Method

There are no specific methods defined for the license type.

Properties

The license type object provides the following properties.

node_count The number of nodes currently in use.

5.3.4 Virtual Object (Virtual)

Virtual object type specifies an implementation module for defining special objects of Kompira, such as processes and incidents.

Field

In the virtual object type, the following fields are defined:

virtual (implementation module) [String] Specifies the path of the Python implementation module of the special object.

Method

Methods specific to virtual object types are not specifically defined.

5.3.5 Job flow (Jobflow)

The job flow type specifies the type of job flow object.

Field

In the job flow type, the following fields are defined.

source [LargeText] The source code string of the job flow.

code [LargeText] The intermediate code character string, as the result of compiling the source of the job flow, is stored. It can not be edited from the browser.

parameters [Dictionary] The intermediate code character string, resulting from compiling the default value of the parameters of the job flow, is stored as a parameter dictionary. Since it is invisible, it can not be edited from the browser.

executable [Boolean] If the job flow can be executed, it is true. If the job flow can not be executed because of a compile error or the like, false is stored. It can not be edited from the browser.

errors [Dictionary] An error message at compile time is stored in a dictionary with the line number of the corresponding source code as a key. It can not be edited from the browser.

compilerVersion [String] Contains the version string of the compiler used to compile the job flow. It can not be edited from the browser.

multiplicity [Integer] Sets the multiplicity of the job flow. If a job flow process that exceeds the number of multiplicity invokes this job flow at the same time, that process is kept waiting until another process completes this job flow call. If the multiplicity is set to a value less than or equal to 0, the multiplicity is interpreted as unlimited.

defaultCheckpointMode [Boolean] Specifies the default checkpoint mode for the job flow.

defaultMonitoringMode [Enum] Specifies the default monitoring mode of the job flow.

Note: If you call a job flow with multiplicity specified, the job flow process acquires the lock. When returning from the job flow call or the job flow ends, the lock is released. Locks can be acquired recursively. Therefore, even if recursively calling a job flow with specified multiplicity, execution of that process will not block.

Multiplicity When another job flow is called at the end in the specified job flow, the acquired lock is released.

Method

Methods specific to the job flow type are not specifically defined.

5.3.6 Channel (Channel)

Channel type specifies the type of channel object. Using channel objects, it is possible to synchronously send and receive messages between different job flow processes.

Field

In the channel type, the following fields are defined.

message_queue [Array] The queue in which messages sent to the channel are stored. It can not be edited from the browser.

event_queue [Array] A queue that stores events waiting to receive messages on a channel. It can not be edited from the browser.

Method

The channel type has the following methods defined.

send [*message*] Sends the message, 'message' to the channel.

Properties

The channel type object provides the following properties.

message_count Indicates the number of messages accumulated in the message queue.

event_count Indicates the number of events accumulated in the event queue.

5.3.7 Wiki page (Wiki)

The Wiki page type specifies the type of wiki page object. Kompira's Wiki page object supports Wiki Creole / Markdown / Textile notation.

Field

In the Wiki page type, the following fields are defined.

wikitext [LargeText] Stores the text of the wiki page.

style [Enum] Select wiki page notation from Creole, Markdown and/or Textile.

Method

Methods specific to Wiki page type are not specifically defined.

5.3.8 ScriptJob

The script job type specifies the type of script job object.

Field

In the script job type, the following fields are defined.

source [LargeText] Stores the source text of the script.

ext [String] Sets the extension of the script. When executing a script on a Windows server, you need to set the extension of the script appropriately.

multiplicity [Integer] Sets the multiplicity of the script job. If a job flow process that exceeds the number of multiplicity invokes this script job at the same time, that process will wait until another process completes this script job invocation. If the multiplicity is set to a value less than or equal to 0, the multiplicity is interpreted as unlimited.

Method

Methods specific to script job types are not specifically defined.

5.3.9 Environment Variables (Environment)

The environment variable type specifies the type of the environment variable object. If an environment variable object is specified in the environment variable section of the user's configuration, when the user executes the job flow, an environment variable dictionary is stored in $ENV so that each value in the dictionary can be referenced from the job flow.

Field

For environment variable type, the following fields are defined.

environment [Dictionary] Stores the environment variable dictionary.

Method

Methods specific to environment variable types are not specifically defined.

5.3.10 Template

The template type specifies the type of the template object.

Field

In the template type, the following fields are defined.

template [LargeText] Stores the template string.

Method

Methods specific to template type are not specifically defined.

Deprecated since version 1.4.7: Changed since version 1.4.7: Please use text type object instead.

5.3.11 Table

Table type specifies the type of table object. A table object, like a directory object, can have multiple child objects. However, the type of the child object is fixed.

Field

For table type, the following fields are defined.

typeObject [Object] Specifies the type of the child object to be stored in this table.

relatedObject [Object (Jobflow or Form)] Specifies the job flow and form that can be executed from the menu of this table. You can now run job flows and forms on selected objects from the table list. In the case of job flow execution, the selected object list is passed to the first parameter of the job flow. For form execution, the object list selected is passed to the objects parameter.

displayList* [Array] Specifies an array of field names of child objects to be displayed in the view of the table.

Method

The table type method is the same as the method provided by the directory type, but the type_obj parameter of the add method is optional.

add [*name*, [*type_obj*, *data*, *overwrite*]] Adds a Kompira object of the type specified in the object type field of the table with the name specified by name under the table. Dictionary type data can be passed to data, so that you can initialize the field value of the object. If you pass true to the overwrite argument, even if an object of the same name exists under the table, it does not cause an error and updates the object.

5.3.12 Management Area (Realm)

The management area type specifies the type of management area object. By defining the management area objects, you can manage managed networks separately for each job manager.

Field

In the management area type, the following fields are defined.

range [Array] Specifies the target range of the network address that this management area has jurisdiction.

disabled [Boolean] If this value is set to true, the management area setting is invalidated.

Method

Methods specific to the controlling area type are not specifically defined.

5.3.13 AttachedFile

Attachment type specifies the type of attachment object.

Field

In the attached file type, the following fields are defined.

attached1 [File] The field containing the first attachment object.

attached2 [File] The field where the second attachment object is stored.

attached3 [File] The third attachment object field is stored.

Method

Methods specific to attachment type are not specifically defined.

5.3.14 Node information (NodeInfo)

Node information type specifies the type of node information object. By specifying the node information object as the __node__ control variable in the job flow, you can specify the target node to execute the command.

Field

In the node information type, the following fields are defined.

hostname [String] Specifies the host name of the node.

ipaddr (IP address) [IPAddress] Specifies the IP address of the node.

port (port number) [Integer] Specifies the port number of the node. If not specified, the default port number corresponding to the connection type is used.

conntype (connection type) [Enum] Selects the connection type of the node from 'ssh' or 'winrs'.

shell [String] Specifies the shell to be used for remote connection. If not specified, '/bin/bash' is used as the default.

use_shell [Boolean] Set to false if you do not want to use the shell when connecting remotely. When connecting to a device that does not have a shell, such as network equipment, it is a good idea to set it to false. The default is false.

proxy [Object] When connecting via SSH via a steppingstone server, specify the node information object to be the platform server. It is used only when connecting with SSH.

account [Object] Specifies account information to be used for remote connection. If explicitly specifying the __account__ control variable of the job flow, that will take precedence.

Method

Methods specific to node information type are not specifically defined.

5.3.15 Account information (AccountInfo)

Account information type specifies the type of account information object. By setting the account information object to the __account__ control variable in the job flow, you can specify account information to be used for remote connection.

Field

In the account information type, the following fields are defined.

user (username) [String] Sets the user name of the account.

password [Password] Sets the password for the account. If an SSH key file with passphrase is set, it is also used as a passphrase.

keyfile (SSH key file) [File] When logging in using the SSH key file, attach the key file.

passphrase (SSH key passphrase) [Password] This is the passphrase you specify for SSH keys with passphrase. If you do not have a passphrase or if it is the same as a password, you can omit it.

Method

Methods specific to account information type are not specifically defined.

5.3.16 Repository (Repository)

The repository type specifies the type of repository object. Using the repository object, you can link with an external VCS repository, you can synchronize data such as pushing data of Kompira's directory to repository, or pull data on repository to the Kompira directory.

Field

In the repository type, the following fields are defined.

URL [URL] Sets the URL of the repository to be synchronized.

repositoryType [Enum] Specify the repository type. In the current version, only 'git' and 'mercurial' are supported. The default is 'git'.

port (port number) [Integer] Specifies the port number to connect to the external repository. If not specified, the default port number is used.

username [String] Specify the user name when connecting to the repository.

password [Password] Specify the password when connecting to the repository.

directory [Object] Specify the Kompira directory to be synchronized.

log [LargeText] The log at the time of synchronous execution is stored.

Method

Methods specific to repository types are not specifically defined.

5.3.17 Mail channel (MailChannel)

Mail channel type specifies the type of mail channel object that imports mail from IMAP4 / POP3 server into the channel.

When the job flow enters the mail reception waiting state from the mail channel, the mail channel fetches mail from the set IMAP4 / POP3 server. If a mail is received, the mail is passed to the job flow and the job flow continues processing. Received mails are deleted from the IMAP4 / POP3 server folder. If the mail folder is empty and can not be received, mail is fetched from the IMAP4 / POP3 server again after the set time elapses with checkInterval. If there is no job flow waiting for the mail channel, polling processing for receiving mail pauses.

Field

In the mail channel type, the following fields are defined.

message_queue [Array] The queue in which the message sent to the mail channel is stored. It can not be edited from the browser.

event_queue [Array] A queue that stores events waiting to receive messages on the mail channel. It can not be edited from the browser.

serverName [String] Sets the host name or IP address of the IMAP4 / POP3 server to be connected.

protocol [Enum] Sets IMAP 4 or POP 3 as the mail reception protocol.

SSL [Boolean] Set to true to communicate via SSL.

port (port number) [Integer] Set the port number of the IMAP server. If not specified, the default port number is used.

mailbox [String] Set up the mailbox to receive. The default is "INBOX". For the POP3 protocol, mailbox settings are ignored.

> **Warning:** You can not set the Japanese mailbox name.

username [String] Set the user name to connect to the IMAP4 / POP3 server.

password [Password] Set the password for connecting to the IMAP4 / POP3 server.

checkInterval [Integer] Specify the interval for checking new messages for IMAP4 / POP3 server in minutes. The default is 10 minutes. When 0 is specified, it becomes the default value. If negative values are set, new messages will not be confirmed.

timeout [Integer] Specify the reception timeout for IMAP4 / POP3 server in seconds. If it is empty or set to 0 or a negative value, timeout does not occur on reception.

disabled [Boolean] Disable the connection to IMAP4 / POP3 server.

log [LargeText] Connection log of IMAP4 / POP3 server is stored.

logSize [Integer] Specify the maximum size of the log. If it exceeds the maximum size, it will be deleted from the old log message.

Method

Methods specific to mail channel type are not specifically defined.

5.3.18 Form

The form type specifies the type of the form object that provides a view of the user input form. The items of the input form can be freely defined by the user.

When the user submits the form, the information entered in the form is submitted as dictionary type data to the specified submission object. If the submitted object is a channel type, the data is placed in the message queue of that channel object. If the submitted object is a job flow, the dictionary data is expanded to the parameters of the job flow and execution starts.

Field

submitObject [Object (Channel or Jobflow)] Specify the object to submit the data entered in the form.

fieldNames (field names column) [Array] Specify the list of field names of the input form as an array. The rules for strings that can be used in field names are the same as the job flow language identifier.

fieldDisplayNames (field display names column) [Array] Specify the list of field display names of the input form as an array. An arbitrary character string can be used for the field display name. The order of array elements must correspond to the columns of field names.

fieldTypes (field types column) [Array] Specify the list of field types of the input form as an array. The order of array elements must correspond to the columns of field names.

Method

Methods specific to form types are not specifically defined.

5.3.19 Settings (Config)

The configuration type specifies the type of configuration object that provides a view of the configuration form. The items of the setting form can be freely defined by the user.

When the user saves the setting form, the information entered in the form is saved in the data property of the setting object as dictionary type data.

Field

fieldNames (field names column) [Array] Specify the list of field names of the setting form as an array. The rules for strings that can be used in field names are the same as the job flow language identifier.

fieldDisplayNames (field display names column) [Array] Specify the list of field display names of the setting form as an array. An arbitrary character string can be used for the field display name. The order of array elements must correspond to the columns of field names.

fieldTypes (field types column) [Array] Specify the list of field types of the setting form as an array. The order of array elements must correspond to the columns of field names.

Properties

Configuration type objects provide the following properties.

data The value of the data dictionary entered in the setting form.

Method

Methods specific to configuration type are not specifically defined.

5.3.20 Library

The library type defines a library implemented in Python that can be called from the job flow.

Field

libraryType [Enum] Specify how to define the library. If 'source' is selected, the string stored in the source text will be loaded as a Python module program. When 'safe_source' is selected, the string stored in the source text is loaded as a safe Python module program. If 'module' is selected, the character string specified in the module path is loaded as a module under kompira.library in the Kompira package. The default value is 'source'. This field can not be edited from the browser.

modulePath [String] Specify the module path of the Python library to be loaded. The field used when libraryType is 'module'. This field can not be edited from the browser.

sourceText [LargeText] Write Python source code.

document [LargeText] The document character string of Python module is stored. Error messages are stored at load error. This field can not be edited from the browser.

executable [Boolean] It is true if the Python module is loaded correctly and can be called from the job flow. If loading fails, it is false. This field can not be edited from the browser.

Method

Methods specific to configuration type are not specifically defined.

Invocation example

For library objects, you can call the defined Python functions from the job flow. For example, define a test_lib object with the following Python program as source text.

Python Program

```
def split(s):
    return s.split():

def hello():
    print('Hello, world!')
```

The job flow calling the function defined in this library is as follows.

```
[str = 'foo bar baz']
-> [result = ./test_lib.split(s)]
-> [./test_lib.hello]
```

When the above job flow is executed, the result variable stores the list [‘foo’, ‘bar’, ‘baz’] of split results. Also, “Hello, world!” Is output to the console of the job flow process.

> **Warning:** When using property names (*Properties*) and method names (*Method*), which are pre-built in Kompira objects such as display_name, update, and delete, in Python function names defined in library objects they cannot be called the same thing.

5.3.21 MailTemplate

The mail template type specifies the type of the mail template object.

Field

In the mail template type, the following fields are defined.

subject [String] Stores the template character string that is the subject of the mail.

body [LargeText] Stores the template character string that is the body of the mail.

Method

Methods specific to the mail template type are not specifically defined.

5.3.22 Text

Text type specifies the type of text object that holds plain text or HTML text.

The text object can display the render view rendered by the template engine by accessing http://<Kompira server>/<text object>.render from the browser.

Note: Jinja 2 is used for the template engine. For the notation of templates, see the Jinja 2 document http://jinja.pocoo.org/docs/dev/templates/

Templates can be imported and inherited by specifying the path of another text object with the 'include' and 'extends' tags.

Field

For text type, the following fields are defined.

text [LargeText] Stores a text string.

ext [String] Specify extension for browser access to display render view. For example, if you specify "html" as the extension, you will see a render view when accessing http://<Kompira server>/<text object path>.html and http://<Kompira server>/<text object path> To access the normal view.

contentType [String] Specify the content type of the text. If the content type specification is omitted, the content type is guessed from the extension. Also, if you omit specifying the extension and specify only the content type, the render view is displayed instead of the normal view even if the browser is accessed without the extension.

context: Object(Environment) Specify the environment variable object as the context to pass to the template. From the template you can refer to the key value of the environment variable as a variable.

Method

The following methods are defined for the text type.

render : Gets the text rendered by the template engine.

Properties

The text type object provides the following properties.

content_type Indicates the estimated content type.

New in version 1.4.7: A new text type has been added.

5.3.23 System information type

System information type defines objects that provide Kompira's system information.

Field

A unique field is not defined for the system information type.

Method

No specific method is defined for system information type.

Properties

The system information type object provides the following properties.

engine_started Kompira Indicates the start date and time of the engine.

server_datetime Indicates the current date and time of the Kompira server.

version Indicates the version number of Kompira.

New in version 1.4.8.post2: A new system information type has been added.

5.4 Special objects

Unlike ordinary objects, special objects are built-in objects that are not specified by Kompira's type object. Each type has its own properties and methods. It does not have a field.

5.4.1 Process

An object representing process information at the time of executing a job flow.

Properties

The fields defined in the process object are as follows.

checkpoint_mode [Boolean] True if the process is running in checkpointing mode, false otherwise. It is a writable property.

children [Array of Process] A list of child processes is stored.

console [String] A character string displayed on the console.

current_job [Object] Stores job flow object or script job object currently being executed by the process. If you call another job flow from the job flow, the value of current_job is changed.

elapsed_time: Timedelta Represents the elapsed execution time of the process.

finished_time [Datetime] The date and time when the process ended.

job [Object] Represents a job flow or script job object that started the process.

monitoring_mode [String] A string representing the monitoring mode of the process. It is a writable property.

String	Monitoring Mode
NOTHING	A mail will not be sent.
MAIL	When the process is finished, a mail will be sent.
ABORT_MAIL	When the process is terminated abnormally (aborted), a mail will be sent.

pid [Integer] Process ID.

parent [Object] The job flow object of the parent process.

schedule [Schedule] When a process is started from the scheduler, the corresponding schedule object is stored.

started_time [Datetime] The date and time when the process started running.

status [String] A string representing the execution state of the process.

String	Execution Status
NEW	New (awaiting start)
READY	Executable
RUNNING	Running
WAITING	Waiting for input/command completion
ABORTED	Abnormal termination
DONE	Finished

step_mode [Boolean] Set to true if the process is running in stepping mode, false otherwise. It is a writable property.

suspended [Boolean] It is true if the process is suspended, false otherwise.

user [User] The process execution user. Only for privileged users, you can change the executing user.

New in version 1.5.0.post1: A new monitoring_mode has been added.

Method

delete Delete the process object.

5.4.2 Process list (/process)

The process list (/process) is an object that holds a list of process objects and is implemented as a virtual object (Virtual).

It is possible to iterate over each process object in 'for' and 'pfor' blocks as follows.

```
{ for p in /process |
    print(p)
}
```

5.4.3 Schedule

The schedule object represents the schedule registered in the scheduler of Kompira.

Properties

day: String Represents the day (1 to 31) on which the schedule is to be executed.

day_of_week: String Represents the day of the week or weekday number on which the schedule is to be executed. 0 (Monday) - 6 (Sunday), or mon, tue, wed, thu, fri, sat, sun.

description: String A character string describing the contents of the schedule is stored.

disabled: Boolean This field indicates invalidation of the schedule. It is true if the schedule is invalid, false if it is valid.

hour: String Represents the schedule execution (0 to 23).

job: Object Stores job flow or script job executed by schedule.

minute: String Represents the minute (0 to 59) to execute the schedule.

month: String Represents the month (1 to 12) for executing the schedule.

name: String A character string representing the name of the schedule.

next_run_time: Datetime If the schedule is valid, the date and time of the next execution will be stored. (Read Only)

parameters: Array of String Strings of parameters passed to the job flow and script are allowed.

user: User Schedule user.

week: String Represents the ISO week number (1 to 53) for executing the schedule.

year: String Represents the year (4 digit number) for executing the schedule.

Note: For the fields that specify execution date and time of the above property, Date and time setting field format *Date and time setting field format* can be used.

Method

delete Delete the schedule object.

5.4.4 Schedule list (/scheduler)

The schedule list (/scheduler) is an object that holds a list of schedule objects and is implemented as a virtual object (Virtual). By using the schedule list in for and pfor blocks, it is possible to process each schedule object iteratively.

Method

The following methods are defined in the schedule list.

add [*name*, *job*, [*parameters*, *datetime*]] In the schedule list, add a schedule that has the name specified by name and executes the job flow or script job specified by job. You can specify as an option a parameter list given when executing job flow or script job as parameters argument. An optional datetime argument can be a date-time type value that indicates the job execution date and time.

5.4.5 User

It is an object representing Kompira's user.

Properties

The properties defined in the user object are as follows.

username [String] Username

first_name [String] The user's first name.

last_name [String] The user's surname.

full_name [String] The user's full name.

mailbox [String] Represents an address in the following format.

```
username <email address>
```

email [String] This is the user's email address. It is a writable property.

environment [Object (Environment)] Environment variable object. It is a writable property.

home_directory [Object (Directory)] It is the user's home. It is a writable property.

groups [Array of Group] Group list to which the user belongs.

enable_restapi [Boolean] Indicates whether to enable the REST API. It is a writable property.

auth_token [String] The user's authentication token. This is a read-only property. It is null when the REST API is invalid.

Method

There are no published methods.

5.4.6 Group

This is an object representing a group of Kompira.

Properties

There are no published properties.

Method

There are no published methods.

CHAPTER

SIX

COORDINATION WITH OTHER SYSTEMS

Author Kompira development team

6.1 Introduction

This document will explain how to transfer data to other systems with Kompira, how to receive data from other systems and the required settings etc.

6.2 Sending events to Kompira

Event information can be sent to Kompira by using the kompira_sendevt command included in *Job manager package* and *Send-Event package*. This section explains the event transmission to Kompira using kompira_sendevt.

The kompira_sendevt script packs the `<keyword>=<value>` pair specified by the argument into a message and sends it to the Kompira server.

```
/opt/kompira/bin/kompira_sendevt [options] [<key1>=<value1> ...]
```

Be careful not to put a space on both sides of '=' connecting key1 and value1. The job flow can refer to the received message as dictionary type data.

If no argument is specified, the standard input is sent as one key to the Kompira server.

6.2.1 Sending events from Windows

By installing *Installation on Windows*, you can send events from Windows using the kompira_sendevt command, similar to as in Linux.

Note: Procedure: When installing *Installation on Windows*, the kompira_sendevt command will be installed in `C:\Kompira\Scripts\kompira_sendevt.exe`.

Also, since the default log directory `/var/log/kompira` is not created on Windows, a warning is displayed when executing the kompira_sendevt command, and a log is output on the standard output.

In order to avoid this, create the following configuration file and save it as kompira.conf.

```
[logging]
logdir=.\
```

Use the `--conf` option of the kompira_sendevt command to read the above file. In addition, specify the Kompira server to which the event is sent with the `--server` option.

```
$ kompira_sendevt --conf=kompira.conf --server=<kompira server> test_key=test_message
```

6.2.2 kompira_sendevt options

The kompira_sendevt command has the following options:

Option	Description
`-c, --config=CONF`	Specify the configuration file. (CONF is the configuration file path) By default, /opt/kompira/kompira.conf is read.
`-s, --server=SERVER`	Specify the IP address or server name of the Kompira server as the destination. It takes precedence over specification of the configuration file.
`-p, --port=PORT`	Specify the port number of the message queue of the destination Kompira server. It takes precedence over specification of the configuration file.
`--user=USER`	Destination Kompira specifies the user name of the message queue. It takes precedence over specification of the configuration file.
`--password=PASSWORD`	Destination Kompira specifies the user name of the message queue. It takes precedence over specification of the configuration file.
`--ssl`	SSL is used to connect the message queue.
`--channel=CHANNEL`	Specify the path on the Kompira file system of the channel to send the message. It takes precedence over specification of the configuration file.
`--site-id=SITE_ID`	Specify Kompira site ID. It takes precedence over specification of the configuration file.
`--max-retry=MAX_RETRY`	Specify the maximum number of times to send an event.
`--retry-interval=` `RETRY_INTERVAL`	Specify the interval between events (in seconds).
`--dry-run`	Does not actually transmit data but displays the transmission contents on the standard output.

6.3 Receive e-mails on Kompira

How to use Kompira_sendevt to handle email contents received by the Kompira server in the job flow.

Note: If you are using an IMAP server, you can handle the contents of the email in the job flow by using the mail channel in alternative to the method described below. For details, see *Mail channel (MailChannel)*.

6.3.1 Setting up Linux

By writing the settings in the/etc/aliases file which is an alias for Sendmail, it is possible to specify execution of an arbitrary command for emails addressed to a specific account on the Kompira server.

The following is a setting for sending mail to kompira_sendevt when an email is sent to the kompira account on the kompira server.

```
kompira:    "|LANG=ja_JP.UTF-8 /opt/kompira/bin/kompira_sendevt --channel=/system/
↪channels/Mail"
```

After writing the above in /etc/aliases, the setting is reflected by executing the following command.

```
% newaliases
```

Note: Depending on your system you may need to use smrsh. In that case, create a symbolic link of the kompira_sendevt command in the smrsh directory.

6.3.2 Kompira settings

kompira_sendevt can send values to arbitrary channels. Here is how to create a dedicated channel to receive mail called /system/channels/Mail.

The following is an example of a job flow in which mail contents are received and the contents are displayed.

```
</system/channels/Mail>
-> [mail = $RESULT]
-> mail_parse(mail)
-> [parsed_mail = $RESULT]
-> print(parsed_mail['Subject'])
-> print(parsed_mail['Body'])
```

By using Kompira's built-in job mail_parse, you can parse mail text in MIME format and handle values in dictionary format.

6.4 Coordinating with monitoring systems

Kompira can coordinate with external monitoring servers such as Zabbix and Nagios. By sending event information to the Kompira message queue (RabbitMQ) from the external system you want to link, you can receive the event from the job flow.

In this section, we will explain how to notify Kompira of occurrence of failure by using Zabbix as an example.

6.4.1 Confirming event transmission and receipt

How to prepare a script for sending event information to Kompira on the server running Zabbix. This section explains how to use kompira_sendevt.

1) **Install the Kompira agent on the Zabbix server**

According to the Kompira manual (*Send-Event package*), install Kompira's event sending package on the server on which Zabbix is running. (If you do not operate the job manager, startup settings of the job manager are unnecessary.)

2) **Change of Configuration File**

Rewrite the /opt/kompira/kompira.conf file on the Zabbix server side.

Specifically, set the IP address of the Kompira server or the host name in the server field of the [amqp-connection] section. Also make sure that the channel item in the [event] section is set to /system/channels/Alert.

3) **Message notification confirmation**

At this point, run kompira_sendevt to check that the event can be notified correctly to the Kompira server. On the Zabbix server side, execute the following command.

```
$ /opt/kompira/bin/kompira_sendevt test=hello
```

Next, log in to Kompira, refer to the page of /system/channels/Alert and check that the number of messages is increasing.

4) **How to receive messages**

Next, read the message that arrived at /system/channels/Alert from the job flow. Please define the following job flow and execute it.

```
</system/channels/Alert> -> [message = $RESULT] -> print(message.test)
```

If "hello" is displayed on the console, it was successful.

6.4.2 Zabbix Settings

Next, configure Zabbix.

Log in to Zabbix, create a new action from the "Set Action" menu and create a new operation of the action in it. The type of operation is a remote command.

For example, the contents of the remote command are as follows.

```
Zabbix server:python /opt/kompira/bin/kompira_sendevt status="{TRIGGER.STATUS}"
  severity="{TRIGGER.NSEVERITY}" hostname="{HOSTNAME}"
  trigger_name="{TRIGGER.NAME}" trigger_key="{TRIGGER.KEY}"
  detail="{TRIGGER.KEY}: {ITEM.LASTVALUE}"
```

Here, we have set up to send dictionary data including the following key to Kompira.

Key name	Content (Value)
status	Trigger status
severity	Severity
hostname	Name of the host where the failure occured
trigger_name	Trigger name
trigger_key	Trigger Key
detail	Event detailed information (combination of trigger key and event value)

After that, we will make settings so that the action registered here will be kicked, with the fault event as a trigger. For details, please refer to the Zabbix manual etc.

6.5 Coordinating with Redmine

As an example of coordination with an external ticketing system, we will explain how to issue tickets to Redmine from Kompira's job flow.

6.5.1 Redmine settings

1) **Enable REST API**

From "Administration" -> "Settings" -> "Authentication", save with "Enable Web service by REST" checked.

2) **Create project**

Select "New Project" from "Administration" -> "Project" and create the project "test".

3) **Setting Priorities**

Set the value to the priority of the ticket in "Administration" -> "Enumeration item". (Eg "high" "medium" "low")

Also, set one as the "default value".

(*) If you do not set the default value, the priority_id value is required when calling the API.

4) **Create new user**

Select "New User" from "Administration" -> "User" and create an arbitrary user.

Log in as the user you created and note the API access key on the "Personal Settings" page.

6.5.2 Issuing a ticket

In order to issue Redmine tickets, we convert the necessary information into json format data and send a POST request to the Redmine URL.

To do that, call dictionary type data to urlopen which is a built-in job of Kompira.

Specifically, you can issue a ticket to Redmine by describing the following job flow.

```
|redmine_server = '192.168.0.1'|
|redmine_key = '1234567890abcdef1234567890abcdef12345678'|
|ticket_title = 'Task from Kompira'|
|project_name = 'test'|

[url = 'http://$redmine_server/issues.json?format=json&key=$redmine_key']
-> [ticket = {issue = {subject = ticket_title, project_id = project_name}}]
-> urlopen(url=url, data=ticket, timeout=60, encode='json')
```

For "redmine_key", set the API access key confirmed in "4. Creating a user".

In addition to the above, you can also include information such as ticket priority, description, person in charge and category.

You can also update / delete tickets, get list of ticket information, etc. For details, refer to the Redmine API specifications.

6.6 Receiving SNMP Traps

How to receive SNMP traps in Kompira's job flow using Linux commands snmptrapd (8) and snmptrap (1).

6.6.1 Environment

	IP Address	OS
Kompira Server	192.168.213.100	CentOS 6.5
SNMP Agent Server	192.168.213.101	CentOS 6.5

6.6.2 Kompira Server Settings

Assume Kompira is installed on the Kompira server.

1) **Install snmptrapd**

```
$ yum install net-snmp
```

2) **Edit /etc/snmp/snmptrapd.conf**

Edit snmptrapd.conf to handle SNMP traps.

```
authCommunity     log,execute,net default
traphandle default    /opt/kompira/bin/kompira_sendevt --channel=/system/channels/
↪snmptrap
```

Here default represents "all OIDs".

3) **Add job flow to Kompira**

Create a "/system/channels/snmptrap" channel and create and execute a job flow that waits for data to this channel.

```
</system/channels/snmptrap> ->
print($RESULT)
```

4) **Start snmptrapd**

```
$ service snmptrapd start
```

6.6.3 Setting up the SNMP agent server

Install snmptrap command

```
$ yum install net-snmp-utils
```

6.6.4 Transmission of SNMP trap

Execute the snmptrap command on the SNMP agent server.

```
$ snmptrap -v 2c -c default 192.168.213.100 '' netSnmp.99999 netSnmp.99999.1 s "hello␣
↪world"
```

If the Kompira server can receive it correctly, the following log is displayed in /var/log/messages.

```
$ tail -f /var/log/messages
Dec 13 16:29:30 kompira-server snmptrapd[6110]: 2012-12-13 16:29:30 <UNKNOWN>
[UDP: [192.168.213.101]:56313->[192.168.213.100]]:#012DISMAN-EVENT-
↪MIB::sysUpTimeInstance = Timeticks: (590254) 1:38:22.54
#011SNMPv2-MIB::snmpTrapOID.0 = OID: NET-SNMP-MIB::netSnmp.99999#011NET-SNMP-
↪MIN::netSnmp.99999.1 = STRING: "hello world"
```

In addition, the following received results are displayed on the console of the job flow process that was running on Kompira.

```
<UNKNOWN>
UDP: [192.168.213.101]:56313->[192.168.213.100]
DISMAN-EVENT-MIB::sysUpTimeInstance 0:0:18:39.04
SNMPv2-MIB::snmpTrapOID.0 NET-SNMP-MIB::netSnmp.99999
NET-SNMP-MIB::netSnmp.99999.1 "hello world"
```

CHAPTER

SEVEN

MONITORING KOMPIRA

Author Kompira development team

7.1 Introduction

This document describes how to monitor the state of Kompira using a monitoring system such as Zabbix.

7.2 Monitoring using Zabbix

This document will explain how to acquire the number of Kompira's running processes and the number of incidents being handled in Zabbix.

There are various ways to monitor Zabbix, but here we will explain monitoring using Zabbix Agent's "UserParameter" function and monitoring method by "external script".

This document introduces the monitoring method using Zabbix 2.4.

7.2.1 Preperation

kompira_jq.sh

Whichever monitoring method you use, use the `kompira_jq.sh` script provided by Kompira.

For monitoring by "external script", execute `kompira_jq.sh` on the Zabbix server and on the Kompira server on which Zabbix Agent is installed for monitoring with, use "UserParameter".

Since `kompira_jq.sh` internally uses the curl and jq commands, please install the necessary packages on the Zabbix server or Kompira server according to the monitoring method so that these can be used.

In the CentOS environment, jq can be installed from the EPEL repository and in the AWS environment from the amzn-main repository.

Note: Since kompira_jq.sh that comes with Kompira 1.4.6 or later has become a version compatible with REST-API, it is incompatible with option specification method and older versions.

Kompira server's host settings

Since both monitoring methods access the Kompira server, it is necessary to register the Kompira server URL and REST API token as Zabbix "macros".

On Zabbix, set the REST API token with the following macro names on the "Host Setting" → "Macro" settings screen of the Kompira server.

Macro	Value
{$KOMPIRA_URL}	Kompira Server's URL
{$KOMPIRA_TOKEN}	REST API token

7.2.2 Monitoring with UserParameter

This is the way Zabbix Agent gathers the value of the monitoring item by executing a preset command for the item specified from the Zabbix server.

Zabbix Agent settings

It is necessary to prepare UserParameter's setting files on the Kompira server where Zabbix Agent is installed. Please copy `userparameter_kompira.conf` to `/etc/zabbix/zabbix_agentd.d`

Please restart the Zabbix Agent when the setup file is ready.

```
$ sudo service zabbix-agent restart
Shutting down Zabbix agent:                                [  OK  ]
Starting Zabbix agent:                                     [  OK  ]
```

Zabbix Server settings

For Zabbix Server, you need to set monitoring items using UserParameter, but you can immediately use standard monitoring items by importing `zbx_kompira_basic_templates.xml`.

A template named Template Kompira Server will be created, so apply this template to the Kompira server you want to monitor.

Monitoring items

The following monitoring items can be used as standard.

Name	Overview
Kompira active incidents	Number of active incidents
Kompira active processes	Number of active processes
Kompira active schedulers	Number of active schedules
Kompira active tasks	Number of active tasks
Kompira jobflows	Total number of job flows
Kompira license remain_days	Number of remaining days of license
Kompira objects	Total number of Kompira objects
Memory usage of kompirad process	Memory usage (kompirad)
Memory usage of kompira_jobmngrd process	Memory usage (kompira_jobmngrd)
Number of kompirad process	Number of processes (kompirad)
Number of kompira_jobmngrd process	Number of processes (kompira_jobmngrd)

7.2.3 Monitoring with external scripts

This is a method of collecting the value of the monitoring item by executing the external script on the Zabbix server.

First, please copy the script provided by Kompira `/opt/kompira/bin/kompira_jq.sh` to the directory where the external script on the Zabbix server is located. By default it is: `/usr/lib/zabbix/externalscripts`.

Number of processes

When monitoring the number of processes using the external script `kompira_jq.sh`, create the Item with the following settings:

Name	`Kompira processes`
Type	External check
Key	`kompira_jq.sh[-s,{$KOMPIRA_URL},-t,{$KOMPIRA_TOKEN},-ac,/process]`
Type of information	Numeric (unsigned)
Data type	Decimal

Number of incidents

To monitor the number of incidents using the external script `kompira_jq.sh`, create an Item with the following settings:

Name	`Kompira incidents`
Type	External check
Key	`kompira_jq.sh[-s,{$KOMPIRA_URL},-t,{$KOMPIRA_TOKEN},-ac,/incident]`
Type of information	Numeric (unsigned)
Data type	Decimal

CHAPTER

EIGHT

KOMPIRA REST API REFERENCE

Author Kompira development team

8.1 Introduction

This document describes how to use REST API.

8.2 Common Features

8.2.1 The end point

The end point of REST API is similar to that of the resource path of a regular Kompira object. That is, the root end point is displayed

```
http[s]://<hostname>/
```

as above.

In order to distinguish access from the browser and an API request, in the Accept header of the HTTP Request

```
Accept: application/json
```

must be included.

Alternatively, you could include: `format=json` in the request string.

8.2.2 User Authentification

The two types of authentication that are allowed are: 1. token and 2. session authentication methods.

While using token authentication, include the token key in the Authorization header of the request as follows:

```
Authorization: Token <token key>
```

Alternatively, you can include the token key in the query string of the HTTP request as shown below.

```
token=<token key>
```

When you enable the REST API of each user on the user setting page, an access token is generated. If you disable the REST API and re-enable it, the token is reinitialized.

8.2.3 Format

Only the JSON format of data is supported.

Datetime data

The date and time data should be in UTC and the following format of (ISO8601) must be used.

```
%Y-%m-%dT%H:%M:%S.%fZ or %Y-%m-%dT%H:%M:%SZ
```

It is acceptable to omit microseconds and seconds when entering the data. If Z (the last UTC directive) is omitted, it will be regarded as (JST) local time and will be internally converted to UTC.

Object data

Object data is represented by the absolute path of the object.

File data

When outputting file data, it becomes key holding dictionary data which displays the name and path, as shown below.

```
{ "name": "<file name>", "path": "<file path>" }
```

When entering the data, enter it as key holding dictionary data, showing name and data as shown below.

```
{ "name": "<file name>", "data": "<file contents string>" }
```

8.2.4 Error

When an error occurs, A HTTP status code will appear indicating as such. When this happens, data indicating the reason for the error will be shown in the HTTP response's content body.

Most error data will be shown in dictionary type including the detail key as follows.

```
{ "detail": "<reason of error>" }
```

In the case of a validation error in which the required data is not included in the request data, the following dictionary data will appear:

```
{ "<field name>": ["<error message>", ...],
  "<field name>": ["<error message>", ...],
... }
```

8.2.5 Paginate

When the list is retrieved, paginated data in the form shown, appears as below.

```
{
    "count": <total number of objects>,
    "next": <next page URL>,
    "previous": <previous page URL>,
    "results": <objects data list>
}
```

When specifying and acquiring a page, include page = <page number> in the query path. If you want to get the last page, specify last as the page number.

The default page size is 25. If you want to change the page size, enter:

```
page_size=<page size>
```

into the query string.

8.2.6 Filtering

To filter by attribute of the object, specify `<attribute-name>=<value>` in the query path.

For example, if you want to get a list of only successfully completed processes, specified as shown below.

```
/process?status=DONE
```

If multiple attributes are specified in the query path as shown below, it is filtering by AND.

```
/app.descendant?display_name=test&owner=root
```

Although the above is filtering by exact matching, detailed filtering conditions can be specified by describing the attribute name followed by a lookup as follows.

```
<attribute-name>__<lookup>=<value>
```

For example, you can filter objects that contain `test` in the display name.

```
/app.descendant?display_name__contains=test
```

The lookup types and filtering method is as follows.

Lookup	Filtering method
exact, iexact	The attribute exactly matches the specified value.
contains, icontains	The attribute contains the specified value.
startswith, istartswith	The attribute starts with the specified value.
endswith, iendswith	The attribute ends with the specified value.
regex, iregex	The attribute matches the specified regular expression.
gt, gte	The attribute is greater than specified value (gt). The attribute is greater than or equal to the value specified (gte).
lt, lte	The attribute is less than specified value (lt). The attribute is less than or equal to the specified value (lte).
in	The attribute is included in the specified values.

In filtering others than virtual objects by attribute value, the lookup that can be specified depends on attribute.

Attribute	Specifiable Lookup
owner	exact, in
abspath	exact, iexact, contains, icontains, startswith, istartswith, endswith, iendswith, regex, iregex
display_name	exact, iexact, contains, icontains, startswith, istartswith, endswith, iendswith, regex, iregex
description	exact, iexact, contains, icontains, startswith, istartswith, endswith, iendswith, regex, iregex
created	exact, gt, gte, lt, lte
updated	exact, gt, gte, lt, lte
type_object	exact, in

In filtering virtual objects by attribute value, the lookup that can be specified depends on the data type of the attribute.

Type of the attribute	Specifiable Lookup
String	exact, iexact, contains, icontains, startswith, istartswith, endswith, iendswith, regex, iregex
Integer	exact, gt, gte, lt, lte
Datetime	exact, gt, gte, lt, lte
Object	exact
User	exact
Boolean	exact

If lookup is not specified, exact is applied.

Object

The attributes that can be used for filtering general objects (other than virtual objects) are as follows.

Attribute name	description (type of attribute) or [available lookup]
owner	Owner (User)
display_name	Display name (String)
description	Description (String)
created	Created date (Datetime)
updated	Updated date (Datetime)
type_object	Type object (Object)

In addition, in situations where type objects are identified, you can also specify filtering by field values.

```
field:<field-name>__<lookup>=<value>
```

For example, specify as follows.

```
/.descendant?type_object=/system/types/Jobflow&field:source__contains=urlopen&
↪field:defaultMonitoringMode=MAIL
```

A situation in which a type object is specified means one of the following.

- The type_object attribute filter specifies a type object.
- The endpoint object is a table type, and the type object is set in the table.

If type objects are not specified, specifying filtering by field value results in an error.

In filtering by field value, the lookup that can be specified differs depending on the data type of the field.

Type of field	Specifiable Lookup
String	exact, iexact, contains, icontains, startswith, istartswith, endswith, iendswith, regex, iregex, in, range
Integer	exact, isnull, gt, gte, lt, lte, in, range
Boolean	exact
Datetime	exact, isnull, gt, gte, lt, lte, range
Object	exact, isnull
File	(same as string)
Array	(same as string)
Dictionary	(same as string)

Process Type (Process)

The attributes that can be used for filtering process objects are as follows:

Attribute name	description (type of attribute) or [available lookup]
job	Job object (Object)
user	Execution user (User)
started_time	Start date and time (Datetime)
finished_time	End date and time (Datetime)
status	Status [exact]
schedule	Schedule object (Obejct)
parent	Parent process (Process)
current_job	Running job object (Object)
suspended	Pause flag (Boolean)
lineno	Running line number (integer)
console	Console (String)

Schedule type Processes (Scheduler)

The attributes that can be used for filtering schedule objects are as follows:

Attribute name	description (type of attribute) or [available lookup]
name	Schedule name (String)
description	Description (String)
user	User type (User)
job	Job object (Object)
month	Month [exact, contains]
day	Day [exact, contains]
week	Week [exact, contains]
day_of_week	Day of the week [exact, contains]
hour	Hour [exact, contains]
minute	Minute [exact, contains]
disabled	Disabled flag (Boolean)

Incident type processes (Incident)

The attributes that can be used for filtering incident objects are as follows:

Attribute name	description (type of attribute) or [available lookup]
name	Incident name (String)
device	Device name (String)
service	Service name (String)
created_date	Created date (Datetime)
closed_date	Completed date and time (Datetime)
status	Status [exact]
owner	Owner (User)

Task type (Task)

The attributes that can be used for filtering task objects are as follows:

Attribute name	description (type of attribute) or [available lookup]
`name`	Task name (String)
`title`	Title (String)
`message`	Message (String)
`status`	Status [exact]
`owner`	Owner (User)
`created_date`	Created date (Datetime)
`closed_date`	Completed date and time (Datetime)

User type (User)

The attribyte that can be used for filtering user objects are as follows:

Attribute name	description (type of attribute) or [available lookup]
`username`	User name (String)
`first_name`	First name (String)
`last_name`	Surname (String)
`email`	E-mail (String)
`last_login`	Last login date and time (Datetime)
`is_active`	Enabled (String)
`home_directory`	Home directory (Object)
`environment`	Environment variable (Object)

Group type (Group)

The attributes that can be used for group object filtering are as follows:

Attribute name	description (type of attribute) or [available lookup]
`name`	Group name (String)

8.3 Accessing Kompira objects

8.3.1 How to get object information

Request

- GET <object path>

Response

```
{
    "id": <object ID>,
    "abspath": <object path>,
    "owner": <username of object owner>,
    "fields": <field data dictionary>,
```

(continues on next page)

(continued from previous page)

```
    "extra_properties": <extra properties>,
    "user_permissions": <user permissions dictionary>,
    "group_permissions": <group permissions dictionary>,
    "display_name": <display name>,
    "description": <desciption>,
    "created": <datetime of object created>,
    "updated": <datetime of object updated>,
    "type_object": <path of type obejct>,
    "parent_object": <path of parent object>
}
```

Field data dictionary and object extended attributes are dictionary data containing keys that vary depending on the object type.

8.3.2 Object Information Updates

Request

- PUT <object path>
- PATCH <object path>

The PUT request replaces the entire data of the object. For a partial object update, use the PATCH request instead.

Request Data

```
{
    "owner": <username of object owner>,
    "fields": <field data dictionary>,
    "user_permissions": <user permissions dictionary>,
    "group_permissions": <group permissions dictionary>,
    "display_name": <display name>,
    "description": <desciption>           # optional
}
```

In the case of a PATCH request, omitting an attribute does not change the value of the object corresponding to that key.

Response Updated Object Data

8.3.3 Adding a new object

When a POST request is sent to a directory object or table object, an object is newly created.

Request

- POST <directory or table's object path>

Request Data

```
{
    "owner": <username of object owner>,  # optional
    "fields": <field data dictionary>,
    "name": <object name>,
    "user_permissions": <user permissions dictionary>,
```

(continues on next page)

(continued from previous page)

```
    "group_permissions": <group permissions dictionary>,
    "display_name": <display name>,       # optional
    "description": <desciption>,          # optional
    "type_object": <path of type obejct>
}
```

Response HTTP 201 Created Response to the Newly Created Object's Data

8.3.4 Deleting an object

You can delete the object by sending a DELETE request to the object path.

Processes, schedules, incidents, tasks, objects can also be deleted in this way, one by one.

Request

- DELETE <object path>

Response If successful, HTTP 204 No Content is returned.

8.3.5 Obtaining a list of children and descendants

You can get a list of directory objects or children and descendant objects of table objects.

Request

- GET <object path>.children # child object list
- GET <object path>.descendant # descendant object list

Response A list of object data will be returned.

Note: For objects other than directories and tables, an empty list is returned.

8.3.6 Executing a job flow

Request

- POST <job flow path>.execute

Request Data

```
{
    "step_mode": <step mode>,                   # true or false
    "checkpoint_mode": <checkpoint mode>,       # true or false
    "monitoring_mode": <monitoring mode>,       # NOTHING, MAIL, ABORT_MAIL
    "parameters": <jobflow parameters dictionary>
}
```

Response The path of the executed job flow process is returned.

8.3.7 Executing a script job

Request

- POST <script job path>.execute

Request Data

```
{
    "node": <path of node object>,
    "account": <path of account object>,
    "command_line": <command line string>
}
```

Response The path of the executed job flow process is returned.

8.3.8 Sending Messages

Messages can be sent to Channel Objects.

Request

- POST <channel object path>.send

Request Data Can be sent in any JSON data format

Response If it succeeds, HTTP 200 OK is returned.

8.3.9 Received Messages

Receiving a message from a channel object

Request

- POST <channel object path>.recv

Request Data

```
{
    "timeout": <timeout (seconds)>
}
```

Response If it succeeds, the received data is returned. When the receive timeout expires, HTTP 408 Request Timeout is returned as the status code.

Note: If there is no data on the channel, wait for the specified number of seconds with timeout. The default value for timeout is 0 seconds.

8.4 Process

8.4.1 Retreiving lists

Request

- GET /process

To retrieve the child process list, use the following request:

- GET /process/id_<process ID>.children

Response A list of process details data will be returned.

8.4.2 Obtaining process detail data

Request

- GET /process/id_<process ID>

Response

```
{
    "id": <process id>,
    "abspath": <path of process object>,
    "user": <username of execution user>,
    "elapsed_time": <elapsed time from process started>,
    "started_time": <datetime of process started>,
    "finished_time": <datetime of process finished>,
    "status": <process status>,
    "exit_status": <exit status>,
    "result": <result>,
    "error": <error>,
    "suspended": <suspended>,
    "lineno": <line number>,
    "console": <console string>,
    "job": <path of starting jobflow>,
    "schedule": <path of schedule>,
    "parent": <path of parent process>,
    "current_job": <path of executing jobflow>
}
```

8.4.3 Process' operation

Request

- POST /process/id_<process ID>.terminate # Cancel process execution
- POST /process/id_<process ID>.suspend # Pause process execution
- POST /process/id_<process ID>.resume # Restart process execution

Request Data You can

```
{
    "step_mode": <step mode>,                    # true/false
    "checkpoint_mode": <checkpoint mode>         # true/false
}
```

Response Returns “true” on success, “false” on failure.

8.4.4 Wait for the completion of execution of the process

Request

- POST /process/id_<process ID>.wait

Request Data

```
{
    "timeout": <timeout value>                  # an integer value of 0 or more
}
```

Response On success, detailed information on the process is returned. When timeout occurs, HTTP 408 Request Timeout is returned as the status code.

8.5 Schedule

8.5.1 Obtaining the schedule

Request

- GET /scheduler

Response A list of schedule detail data is returned.

8.5.2 Obtaining Schedule details

Request

- GET /scheduler/id_<schedule ID>

Response

```
  {
      "id": <schedule id>,
      "abspath": <path of schedule object>,
      "user": <username>,
      "scheduled_datetimes: <datetimes of scheduled>,
      "parameters": <parameters list>,
      "name": <schedule name>,
      "description": <description>,
      "year": <year>,
      "month": <month>,
      "day": <day>,
      "week": <week>,
      "day_of_week": <day of week>,
      "hour": <houe>,
      "minute": <minute>,
      "disabled": <disabled>,
      "job": <path of execution jobflow>
}
```

8.5.3 Schedule update

Request

- PUT /scheduler/id_<schedule ID>
- PATCH /scheduler/id_<schedule ID>

Request Data

```
{
    "user": <username>,
    "parameters": <parameters list>,
    "name": <schedule name>,             # mandatory
    "description": <description>,
    "year": <year>,
    "month": <month>,
    "day": <day>,
    "week": <week>,
    "day_of_week": <day of week>,
    "hour": <houe>,
    "minute": <minute>,
    "disabled": <disabled>,
    "job": <path of execution jobflow>   # mandatory
}
```

Response Updated Object Data

8.5.4 Creating a schedule

Request

- POST /scheduler

Request Data Same as the update request

Response Data of the created object

8.6 Incident

8.6.1 Obtaining the incident list

Request

- GET /incident

Response A list of incident detail data is returned.

8.6.2 Obtaining incident details

Request

- GET /incident/id_<Incident ID>

Response

```
{
    "id": <incident id>,
    "abspath": <path of incident object>,
    "owner": <username of owner>,
    "worklogs": <worklogs>,
    "alerts": <alerts>,
    "name": <incident name>,
    "device": <device name>,
    "service": <service name>,
    "created_date": <datetime of incident created>,
    "closed_date": <datetime of incident closed>,
    "status": <incident status>               # "OPENED", "WORKING", "CLOSED"
}
```

8.6.3 Updating Incidents

Request

- PUT /incident/id_<incident ID>
- PATCH /incident/id_<Incident ID>

Request Data

```
{
    "owner": <username of owner>,
    "name": <incident name>,
    "device": <device name>,
    "service": <service name>,
    "status": <incident status>
}
```

Response Updated Object Data

8.6.4 Adding a work log

Request

- POST /incident/id_<incident ID>.worklogs

Request Data

```
{
    "user": <username>,
    "description": <description>
}
```

Response Data of the added work log

8.6.5 Creating a incident

Request

- POST /incident

Request Data Same as the update request

Response Data of the created object

8.7 Task

8.7.1 Obtaining a list of tasks

Request

- GET /task

Response List of task detail data

8.7.2 Obtaining task details

Request

- GET /task/id_<Task ID>

Response

```
{
    "id": <task id>,
    "abspath": <path of task object>,
    "owner": <username of owner>,
    "assigned_users": <assigned users>,
    "assigned_groups": <assigned groups>,
    "name": <task name>,
    "title": <task title>,
    "message": <message>,
    "action_text": <action text>,
    "result": <result>,
    "status": <status>,          # "WAITING", "ONGOING", "DONE", "CANCELED"
    "created_date": <datetime of task created>,
    "closed_date": <datetime of task closed>
}
```

8.7.3 Cancelling a task

Request

- POST /task/id_<incident ID>.cancel

Response If it succeeds, HTTP 200 OK is returned.

8.7.4 Submitting a task

Send a message to the task channel.

Request

- POST /task/id_<incident ID>.submit

Request Data

```
{
    "result": <result message>
}
```

If request data is omitted, “OK” is transmitted to the task channel.

8.8 User / Group Management

8.8.1 Obtaining a user list

Request

- GET /config/user

Response List of user detail data

8.8.2 Obtaining User Details

Request

- GET /config/user/id_<user ID>

Response

```
{
    "id": <user id>,
    "abspath": <path of user object>,
    "groups": <groups>,
    "last_login": <datetime of last login>,
    "username": <username>,
    "first_name": <first name>,
    "last_name": <last name>,
    "email": <E-mail address>,
    "is_active": <active flag>,
    "home_directory": <path of home directory>,
    "environment": <path of environment object>
}
```

8.8.3 Updating Users

Request

- PUT /config/user/id_<user ID>
- PATCH /config/user/id_<user ID>

Request Data

```
{
    "groups": <groups>,
    "last_login": <datetime of last login>,
    "username": <username>,             # mandatory
    "password": <password>,             # mandatory
    "first_name": <first name>,
    "last_name": <last name>,
    "email": <E-mail address>,
    "is_active": <active flag>,
    "home_directory": <path of home directory>,
    "environment": <path of environment object>
}
```

Response Updated Object Data

8.8.4 Create new users

Request

- POST /config/user

Request Data Same as for Updating Users

Response Data of the created object

8.8.5 Obtaining the group list

Request

- GET /config/group

Response List of group detail data

8.8.6 Obtaining group details data

Request

- GET /config/group/id_<group ID>

Response

```
{
    "id": <group id>,
    "abspath": <path of group object>,
    "name": <group name>
}
```

8.8.7 Updating Groups

Request

- PUT /config/group/id_<group ID>
- PATCH /config/group/id_<group ID>

Request Data

```
{
    "name": <group name>
}
```

Response Updated Object Data

8.8.8 Adding a new group

Request

- POST /config/group

Request Data Same as for updating groups

Response Updated Object Data

INDEX