Matahari Developer's Guide

by the Matahari Team

June 15, 2011

Contents

Abstract

This document is intended for developers who wish to implement an enterprise messaging framework within their organization. The main focus of this article is the fundamentals behind Matahari & QMF.

1 Getting Started

1.1 Requirements

1.1.1 Packages

Packages to install for source based compilation and rpm building

```
projects/> yum groupinstall "Fedora Packager" "Development Libraries" "Development ↩
    Tools" "MinGW cross-compiler"
```

Optional packages may include: mingw32-cxxtest

1.1.2 Checkout Source Code

```
projects/> git clone git://github.com/matahari/matahari.git
```

1.1.3 Develop against Fedora packages

```
projects/> yum install matahari* mingw32-matahari*
```

1.2 Verify Matahari Builds

It's always a good idea to build and test the existing code to make sure no problems crop up before attempting to develop an agent. There are 2 commands provided for cross compilation.

1.2.1 Building for Linux platform

```
$ make linux.build
```

1.2.2 Building for Windows platform

```
$ make windows.build
```

Once building is complete our unittest's should run automatically on the Linux platform.[1] Before beginning the journey to create an agent make sure that these core unit test **never fail**.

1.3 Verify our coding guidelines

See `https://github.com/matahari/matahari/wiki/Coding-Guidelines`

[1] Windows builds will need their unittests run on a Windows host.

2 Preparing Agent

2.1 The End Result

What is it exactly the agent in question is going to be responsible for? Once that is defined an API needs to be outlined. For simplicities sake the API will only consist of generating a report when an agent is disconnected from the broker. A simulation of a crash can be considered when an agent unintentially disconnects.

2.2 Creating the API

All public API routines are kept within **matahari/src/include/matahari/** and **crashreporter.h** will be the header file used for defining our API.

2.2.1 Create the routines that will be publicly accessible.

Listing 1: crashreporter.h

```
/* crashreporter.h - Copyright (C) 2011 Red Hat, Inc.
 * Written by Adam Stokes <astokes@fedoraproject.org>
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public
 * License as published by the Free Software Foundation; either
 * version 2.1 of the License, or (at your option) any later version.
 *
 * This software is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
 * General Public License for more details.
 *
 * You should have received a copy of the GNU General Public
 * License along with this library; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
 */

/**
 * \file
 * \brief Crashreporter API
 * \ingroup coreapi
 */

#ifndef __MH_CRASHREPORTER_H__
#define __MH_CRASHREPORTER_H__

extern void mh_report_crash(const char *address, const char *reason);

#endif // __MH_CRASHREPORTER_H__
```

mh_report_crash is the routine that will be accessed by the broker when the agent disconnected.[2]

[2]All public API routines are prefixed with mh_

2.3 Schema Definition

A schema describes the structure of management data. Each agent provides a schema that describes its management model including the object classes, methods, events, etc. that it provides. In the current QMF distribution, the agent's schema is codified in an XML document. In the near future, there will also be ways to programatically create QMF schemata.[3]

CrashReporter schema will be located alongside our agent code in the folder **matahari/src/crashreporter/**[4]

Create the **schema.xml** file with the following contents

Listing 2: schema.xml

```
<schema package="org.matahariproject">

    <!-- CrashReporter API -->
    <class name="CrashReporter">
        <property name="uuid"           type="sstr" access="RO" desc="Host UUID" />
        <property name="hostname"       type="sstr" access="RO" desc="Hostname" index↩
             ="y"/>

        <statistic name="qmf-gen-no-crash" type="absTime" desc="Dummy stat to stop ↩
             qmf-gen from crashing." />

        <method name="report_crash"          desc="Report crash"/>
          <arg name="address"                dir="I" type="sstr" />
          <arg name="reason"                 dir="I" type="sstr" />
        </method>
    </class>
</schema>
```

2.4 Build Instructions with cmake

The build tool of choice is CMake. Setting up our agent to be included into the distribution should contain a **CMakeLists.txt** which gives the build process the necessary instructions.

Listing 3: CMakeLists.txt

```
set(BASE "crashreporter")
set(BASE_LIB "m${BASE}")
set(QMF_AGENT "matahari-qmf-${BASE}d")

# QMF daemon
if(WITH-QMF)
    set(SCHEMA_SOURCES ${CMAKE_CURRENT_BINARY_DIR}/qmf/org/matahariproject/↩
         QmfPackage.cpp)
    generate_qmf_schemas(${CMAKE_CURRENT_SOURCE_DIR}/schema.xml ${SCHEMA_SOURCES})
    include_directories(${CMAKE_CURRENT_BINARY_DIR})

    add_executable(${QMF_AGENT} ${BASE}-qmf.cpp ${SCHEMA_SOURCES})
```

[3] Taken from https://cwiki.apache.org/qpid/qpid-management-framework.html#QpidManagementFramework-Schema

[4] This is where the main agent code resides

```
    target_link_libraries(${QMF_AGENT} ${BASE_LIB} mcommon_qmf)

    if(NOT WIN32)
        configure_file(${CMAKE_SOURCE_DIR}/matahari.init.in ${↩
            CMAKE_CURRENT_BINARY_DIR}/matahari-${BASE})
        install(FILES ${CMAKE_CURRENT_BINARY_DIR}/matahari-${BASE} DESTINATION ${↩
            initdir})
    endif(NOT WIN32)

    install(TARGETS ${QMF_AGENT} DESTINATION sbin)
endif(WITH-QMF)
```

3 Creating the Agent

3.1 Agent initialization code

Listing 4: crashreporter-qmf.cpp

```
/* crashreporter-qmf.cpp - Copyright (C) 2011 Red Hat, Inc.
 * Written by Adam Stokes <astokes@fedoraproject.org>
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public
 * License as published by the Free Software Foundation; either
 * version 2.1 of the License, or (at your option) any later version.
 *
 * This software is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
 * General Public License for more details.
 *
 * You should have received a copy of the GNU General Public
 * License along with this library; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
 */

#ifndef WIN32
#include "config.h"
#endif

#include "matahari/mh_agent.h"

#include "qmf/org/matahariproject/QmfPackage.h"

#include <qpid/agent/ManagementAgent.h>

extern "C" {
#include <stdlib.h>
#include <string.h>
#include <glib.h>
#include <glib/gprintf.h>
#include "matahari/crashreporter.h"
#include "matahari/logging.h"
#include "matahari/network.h"
#include "matahari/host.h"
#include <sigar.h>
#include <sigar_format.h>
};

class CrashReporterAgent : public MatahariAgent
{
private:
    qmf::org::matahariproject::PackageDefinition _package;
public:
    virtual int setup(qmf::AgentSession session);
    virtual gboolean invoke(qmf::AgentSession session, qmf::AgentEvent event,
                            gpointer user_data);
};

int
main(int argc, char **argv)
{
    CrashReporterAgent agent;
    int rc = agent.init(argc, argv, "CrashReporter");
    if (rc == 0) {
        agent.run();
    }
    return rc;
}

int
CrashReporterAgent::setup(qmf::AgentSession session)
{
    _package.configure(session);
    _instance = qmf::Data(_package.data_Postboot);

    _instance.setProperty("hostname", mh_hostname());
    _instance.setProperty("uuid", mh_uuid());
```

```
    _agent_session.addData(_instance, "crashreporter");
    return 0;
}

gboolean
CrashReporterAgent::invoke(qmf::AgentSession session, qmf::AgentEvent event, ↩
     gpointer user_data)
{
    if (event.getType() != qmf::AGENT_METHOD) {
        session.methodSuccess(event);
        return TRUE;
    }

    const std::string& methodName(event.getMethodName());

    if (methodName == "report_crash") {
            mh_crashreport(event.getArguments()["address"].asString().c_str(),
                           event.getArguments()["reason"].asString().c_str());
        }
    } else {
        session.raiseException(event, MH_NOT_IMPLEMENTED);
        goto bail;
    }

    session.methodSuccess(event);

bail:
    return TRUE;
}
```

3.2 Writing Test Cases

3.3 Testing the Agent

3.3.1 API level test with CxxTest

3.3.2 Functional testing with Beaker

4 Finalizing the Agent

4.1 Build

4.2 Using Mock

4.3 Source Building

4.4 Publish

Those wishing to publish their agents for the general public can follow the below link which describes necessary steps in becoming a Fedora contributor and submitting the package for inclusion into the next release.

Visit
`https://fedoraproject.org/wiki/Join_the_package_collection_maintainers#How_to_join_the_Fedora_Package_Collection_Maintainers.3F`