Voyager OSGi Integration Developer's Guide

Version 1.0 for Voyager 8.0

Table of Contents

Introduction

Demonstrating Voyager operation as an OSGi™[1] bundle highlights the unique features of each platform. Voyager brings to OSGi transparent access to OSGi services and OSGi provides Voyager operational flexibility.

Preface

This manual provides detailed information about the features available in Voyager for Voyager and OSGi integration. This guide assumes basic knowledge of distributed computing concepts, familiarity with the Java language or a .NET programming language, and general knowledge of Voyager and the OSGi R4.0 Specification.

The examples illustrated have been written and tested using the Equinox OSGi framework implementation, however, the Voyager OSGi integration code only uses OSGi interfaces and field values defined in the OSGi Specificaiton Release 4.0.

This preface covers the following topics:

- Definitions

- Voyager installation and development requirements

- Voyager installation directories

- Deploying Voyager applications

- Contacting technical support

Definitions

JME   — *Java Micro Edition.* In reference to running Voyager this term implies a supported version and configuration for the JME.

JSE   — *Java Standard Edition.* In reference to running Voyager this term implies a supported version and configuration for the JSE.

.NET   — *Microsoft .NET Framework.* In reference to running Voyager this term implies a supported version and configuration for the JSE.

---

[1] OSGi is a trademark or a registered trademark of the OSGi Alliance in the United States, other countries, or both.

VM   — *Virtual Machine.* This term refers generically to a supported Voyager execution environment – either a Java Virtual Machine or a .NET Common Language Runtime environment.

Voyager OSGi Installation and Development Requirements

Running Voyager as an OSGi bundle you to have:

- An installation of Voyager, including all Voyager's prerequisites.

- An OSGi framework such as the Equinox framework that ships with the Eclipse development tool.

- Optionally, an installation of Ant 1.7 or newer, the build tool distributed by the Apache project. The Voyager OSGi examples contain Ant scripts for building and executing the bundles. Ant is not needed if you don't plan to run the Voyager OSGi examples.

- Optionally, an installation of the Sun Sun Java Wireless Toolkit for CLDC, needed to build and execute the OSGi examples that show clients running in a CLDC emulation environment accessing an OSGi framework running on Java SE.

Voyager OSGi Installation Directories

The directory structure of Voyager OSGi follows. The paths are relative to the Voyager install directory. In an Unix-like environment the "\" becomes "/".

`platform\cldc\lib\osgi\ve-osgi-cldc.jar`	The OSGi extension to ve-core-cldc.jar.
`platform\cldc\lib\osgi\`	The jar files supporting OSGi in JSE.
`examples\java\cldc\lib\ve-osgi-cldc.jar`	The jar file containing the CLDC portion of the prebuilt CLDC to JSE integration example.
`examples\java\cldc\java\examples\osgi`	The CLDC source code for the CLDC to JSE integration example.
`examples\java\se-cdc\java\examples\osgi`	The source code for the JSE examples.

Contacting Technical Support

Recursion Software welcomes your problem reports and appreciates all comments and suggestions for improving Voyager. Please send all feedback to the Recursion Software Technical Support department.

Technical support for Voyager is available via email and phone. You can also contact Technical Support by sending email to psupport@recursionsw.com or by calling (972) 731-8800.

Note: When submitting an issue via email, if you have a Customer Support ID be sure to include it on the first line of the message body.

Voyager and OSGi Integration Strategy

Comparison: OSGi vs. Voyager.

The features provided in a Voyager framework have many overlapping responsibilities with an OSGi framework, however a comparison will help to establish where they diverge and compliment each other.

The OSGi Service Registry vs. The Voyager Service Registry

The OSGi Service Registry

The OSGi Service Registry provides a cooperative model that allows bundles within the same framework instance to share objects between bundles.

A service registered in the OSGi Service Registry is a set of Java objects that can represent server content or a real-world object such as a nearby Bluetooth phone.

The Voyager Service Registry

The Voyager Service registry is managed completely by a Voyager instance.

It is not federated and exists for the sole purpose of managing the Voyager Services configured for that Voyager instance.

Voyager Proxies vs. The OSGi Wire Admin Service

The OSGi Service Registry

The Wire Admin Service is basically a switchboard that de-couples "producer" services from "consumer" services. The producer service is aware of the data produced and the consumer is aware of the data consumed but neither is aware of each other.

The services must implement a Producer or Consumer interface. A Wire interface is implemented by the Wire Admin Service and serves as a link between the producer and consumer. Updates on the wire can be managed by polling or on demand. The Wire's scopes define the different type(s) that can be transferred over the single Wire instance.

Integration with the Wire Admin Service is used at the time of deployment rather than the time of design.

Voyager Proxies

Voyager Proxies are generated dynamically and/or statically (depending on the platform(s) capabilities) to facilitate method invocations on an instance between two different Voyager instances (possibly on different platforms).

The scope of the proxy invocation is defined naturally by the scope of the interface/class including inherited methods. Depending on the way the object is used in the interface, it may have to implement the ISerializable interface or the developer would use the pgen/igen tools in Voyager at design time to facilitate serialization and object invocation. The Voyager framework manages the protocol required for successful transmission.

On a slightly similar note, Voyager proxies are very different from the proxies implemented for the URL Stream and Content handlers.

- OSGi requires proxies to the java.net.URLStreamHandler and the java.net.ContentHandler because the actual instances are cached by the java runtime and cannot be uninstalled (as required by OSGi lifecycle management). The OSGi proxy handles an error condition that arises when the underlying Stream Handler Service becomes unavailable.

- Voyager proxies are managed by Voyager to facilitate method invocation between remote Voyager instances. If a participating Voyager instance terminates, the proxy returns an error when invoked. The Voyager proxy does not natively handle error conditions when a dependent OSGi service is uninstalled. Managing that error condition is the responsibility of the referent implementation of the proxy.

Platforms

The OSGi specification is only relevant to platforms that support Java applications. Implementing the OSGi specification on a .NET application is not trivial since the .NET class loader does not allow one to unload assemblies and perform inter-application domain communication. (see http://www-adele.imag.fr/Les.Publications/intConferences/CCNC2006Esc.pdf)

The Voyager framework is interoperable between Java and .NET platforms. Deploying Voyager within an OSGi Bundle facilitates interop between that OSGi instance and all platforms available to the Voyager instance.

Security

The OSGi framework employs a security module based on the Java 2 security module.

The Voyager framework is a development environment that can be extended to integrate with a client's specific security requirements. Voyager's OSGi Façade for remote management is subject to the OSGi security model for the bundle in which it is deployed.

Benefits of deployingVoyager inside an OSGi Framework

- Voyager can be used to facilitate OSGi interaction on non-OSGi compliant platforms such as .NET.
  - Executing Voyager inside an OSGi bundle extends the lightweight gateway nature of both frameworks.
- Voyager can also be used to efficiently and intuitively facilitate OSGi interaction on remote platforms via proxy invocation.
  - Interaction between Voyager bundles or Voyager applications do not need to manage complicated OSGi wire configurations.
  - The scope of such method interaction is defined by the class structure which implements the invocation.
- Voyager OSGi integration does not compete against an existing OSGi framework installation.
  - Voyager integrates with OSGi using the OSGi specification R 4.0.
  - The VoyagerOSGI bundle can be deployed in any OSGi framework instance that supports the R4.0 specification.

Features

Deploying Voyager Inside An OSGi Instance

Using the AbstractVoyagerActivator

The `AbstractVoyagerActivator` class, included in `ve-osgi-se-voyagertestbundle.jar` in package `com.recursionsw.ve.osgi.activator` and in source code as `examples.osgi.common.AbstractVoyagerActivator` in the `examples\java\se-cdc\java` examples directory.

Both the Bundle Admin Server (see `examples\java\se-cdc\java\examples\osgi\bundleAdminServer`) and the Yellow Pages (see `examples\java\se-cdc\java\examples\osgi\yellowpages`) examples make use of the `AbstractVoyagerActivator` class.

The `AbstractVoyagerActivator` class implements the `org.osgi.framework.BundleActivator` interface, including final implementations of the `start(BundleContext)` and `stop(BundleContext)` methods. The source code comments provide additional usage details, including use of the `start` and `stop` methods' `before` and `after` methods.

Integration with the IBundleLocationRegistry

The `IBundleLocationRegistry` is a persistent map that associates a Bundle's Symbolic Name to its URL of the jar file for that Bundle. The IBundleLocationRegistry is referenced to determine the URL of a bundle before it is known by the OSGi Service Registry. In this manner, the local OSGi framework instance can manage the locations of Bundles prior to being installed in the OSGi framework instance. Similar to the `AbstractVoyagerActivator` class, this interface exists in both the `com.recursionsw.ve.osgi.activator` and `examples.osgi.common` packages. An example implementation of the interface is found in `examples.osgi.bundleAdminServer.BundleAdminLocationRegistry`, resident in the Bundle Admin Server example.

The OSGi Admin Façade

Voyager implements `com.recursionsw.ve.osgi.facade.pub.IOSGIAdminFacade` to make bundle management available to an application not running in the framework. In particular, by using a Voyager proxy for `IOSGIAdminFacade`, a remote client can manage a framework's bundles. The interface provides methods on the OSGi framework the implement the following capabilities.

- Querying the framework for a list of registered bundles.
- Querying the framework for a list of services.
- Installing and uninstalling a bundle.
- Starting, stopping, and refreshing a bundle.
- Retrieving a bundle's state.
- Adding and removing listeners for framework events, bundle events, and service events.
- Querying the state of Voyager's OSGi yellow pages service.

For additional details, including the method signatures and supporting types, please refer to the JavaDoc for `IOSGIAdminFacade`.

Voyager OSGi Examples

Overview

This chapter describes examples illustrating the basic means to integrate Voyager with OSGi on various platforms.

In this chapter, you will learn to:

- Edit the build properties to reflect your computer's configuration

- Build the examples using the provided Ant build scripts

- Execute the examples using the provided Ant build scripts

In each example's source directory is a `build.xml` file, which is an Ant build script, a text file named `properties.xml.example`, and a `README`.txt file. The normal procedure is to first review the `README`.txt file. The next step is to copy `properties.xml.example` to `properties.xml` and edit the system-dependent properties. Running Ant without command line parameters typically builds the example. Executing `ant -p` will display a description of a `build.xml` file, including a list of public targets.

The Bundle Admin Server

This section illustrates key aspects regarding design, compilation and deployment of the `bundleAdminServer.jar`.

The activator of the `bundleAdminServer.jar` uses the `AbstractVoyagerActivator`. The purpose of the `AbstractVoyagerActivator` is to provide a simple integration point between Voyager and OSGi. The `AbstractVoyagerActivator` is used to wrap a Voyager instance inside an OSGi bundle. By this, Voyager is provided access to the OSGi BundleContext; the interface to the OSGi framework instance. The `AbstractVoyagerActivator` has a flexible design aimed at utilizing the strengths of both frameworks.

The example source can be found in `examples/osgi/BundleAdminServer`. This package bundle will:

- Start Voyager inside an OSGi bundle.

- Provide the default private implementation for the `IOSGIAdminFacade` interface.

- Advertise the `IOSGIAdminFacade` interface in Voyager's Namespace service.

In this package are several key classes:

- The `BundleAdminServer` class extends the `AbstractVoyagerActivator` and uses the default implementation for the `IOSGIAdminFacade` class (defined in `ve-osgi-se-facade-impl.jar`).

- The `BundleAdminLocationRegistry` is a special implementation of the `com.recursionsw.ve.osgi.activator.IBundleLocationRegistry` for the use of this example.

    o This class is used on the server side to provide a URL reference to `ve-osgi-simplebundle.jar` file. This bundle, called the `SimpleBundle`, exists so it can be administered by the examples. It was provided in the `bundleAdminServer.jar` to guarantee that this bundle will be available when executing the examples.

This package also contains a `build.xml` used to construct the `bundleAdminServer.jar`, start the Equinox framework and deploy the `bundleAdminServer.jar` into the Equinox instance. The use of Equinox is for demonstration purposes only. The `bundleAdminServer.jar` is capable of being deployed and started in any OSGi framework instance that is compliant with the OSGi Specification Release 4, Version 4.1. The next section describes aspects of the `build.xml` script in more detail.

Compiling The Bundle Admin Server

The `build.xml` file used to compile and deploy the remote bundle server is located in `examples/osgi/bundleAdminServer/build.xml`.

The build.xml file uses a `properties.xml` as an interface to the developer's environment. Before attempting to use the build.xml, create a `properties.xml` file based on the contents of the `properties.xml.sample`.

The `README.txt` file also contains fine-grain details about the example and the build process. The following sections highlight several aspects of the build.

Required Input Files

The following files are required input to build the bundle admin server.

- `ve-osgi-simplebundle.jar`

- This jar contains the `SimpleBundle` instance. It is provided in the `bundleAdminServer.jar` to guarantee that this bundle will be available when executing the examples. Note that while the examples have been coded to administer to this `SimpleBundle`, the examples can administer over any valid bundle for that OSGi instance.

- `ve-core.jar`

- This jar contains all the essential core components to use Voyager on a JavaSE platform.

- `ve-osgi-se-facade-activator.jar`

  - This jar contains the activator and public interfaces for the admin facade.

- `ve-osgi-se-facade-impl.jar`

  - This jar contains the server-side (default) implementation for the façade interface.

- `VoyagerOSGI.properties`

  - This file contains all the properties required for remote bundle admin clients to connect to the instance of the bundle admin server.

Required Output Files

The build process creates the following artifacts:

- `BundleAdminServer.jar`

  - This is the OSGi bundle that deploys the Voyager instance in the OSGi framework and exposes the admin interface for use by the client application examples.

- `ve-osgi-bundleAdminServer-client-public.jar`

- This jar file contains the public interfaces advertised by the `bundleAdminServer`. The build process derives the public interfaces from the `ve-osgi-client-public.jar` file and inserts the properties file defined in the `examples.osgi.bundleAdminServer.VoyagerOSGI.properties` file. This jar file is required by the build process of any client java application that intends to interact with the example remote server.

  - The `VoyagerOSGI.properties` file contains crucial properties required to allow any Voyager client application to connect to the respective Voyager server deployed an OSGi instance. On the server side, the `AbstractVoyagerActivator` looks for this properties file if the instance advertises the `IOSGIAdminFacade` interface. For the bundleAdminServer example, the `examples.osgi.bundleAdminServer.VoyagerOSGI.properties` file contains the properties required by the example client applications. Note

that the `build.xml` could create this file on the fly, however, it was included as a separate file to simplify the example.

The `VoyagerOSGI.properties` file can contain any *(name,value)* specifications but *must* contain 3 essential properties. These three essential properties are:

- VoyagerSE-OSGI-URL

    o Used to define the URL of the VoyagerSE instance launched inside the Voyager Bundle.

- VoyagerSE-OSGI-FacadeName

    o Used to define the name of the façade instance in the VoyagerSE namespace.

- VoaygerSE-OSGI-BundleName

    o Used to define the name of the VoyagerSE bundle.

Running The Bundle Admin Server

The `build.xml` file contains a target to start the *adminBundleServer.* This target will launch the Eqninox OSGi framework instance, and start the *adminBundleServer*.

After launching the server, a Voyager instance will be started inside the OSGi framework and all example clients that use this server can be executed.

The following snippit illustrates the output of a successful deployment.

```
[VoyagerActivator] VoyagerOSGIFacade bound in Namespace to
"VoyagerOSGIFacade"
[BundleAdminServer] Voyager SE started as a server at //:9010
[BundleAdminServer] Voyager Bundle [BundleAdminServer]  started.
```

Based on the output, the Voyager SE instance started at the URL `//:9010`, the admin façade is advertised in the Voyager namespace under the name, `VoyagerOSGIFacade` and the OSGi bundle has been started with the symbolic name `BundleAdminServer_1.0.0.`

The Bundle Admin Client

This section illustrates key aspects regarding design, compilation and deployment of the Voyager client application that reference the remote bundle management capabilities of

the bundle admin server. Please reference the section in the examples titled, **The Bundle Admin Server** for more information about the bundle admin server.

The bundle admin client illustrates how a platform can derive the utility of the OSGi framework without the native support of the OSGi framework on the client's platform. At the time this document is written, the CLDC and .NET platforms do not support the OSGi framework, however, through the integration with Voyager, the examples illustrate how the CLDC platform can be used to manage bundles and listen for various events from the instance.

The CLDC bundle admin client is provides an example that can be easily created on the .NET platform using Voyager. Please consult the Voyager examples for the .NET platform as a guide to interoperate between JavaSE and .NET.

Design of The Bundle Admin Client

The examples source can be found in the `examples.osgi` and `examples.osgi.ui` packages.

The OSGi examples on the CLDC platform use the MVC (Model-View-Controller) pattern where the Model is the example and the Controller and View define the behavior and the visual aspects, respectively. All examples implement the `examples.common.IExample` interface and obtain all resources through the singleton `examples.common.ExampleContext`. In this way, all example code is isolated for clarity.

The `examples.osgi.VoyagerCLDCOSGIExamples` instance is the entry point for bundle admin client. It is responsible for initializing an instance of the `VoyagerOSGICLCExamplesConfigurationManager` and mapping the relationship between the various examples and the controllers that have been designed for the UI.

All examples depend on the startup logic in the `examples.osgi.VoyagerOSGIExamplesConfigurationManager` class. The `startup()` method:

- StartsVoyager on the CLDC platform

- Establishes a factory instance required to create a single dimensional array of type `BundleReference`

- Looks up the proxy to the `IOSGIAdminFacade` in the namespace of the Voyager bundle.

- Places the instance of the `VoyagerOSGIExamplesConfigurationManager` in the `examples.osgi.ExampleContext` singleton so it can be referenced by the examples to obtain the instance of the `IOSGIAdminFacade` and other configuration properties obtained through the MIDLet instance.

- The primary examples screen provides a list of examples to the user. The map returned by the method `VoyagerCLDCOSGIExamples.buildExampleMap()` creates a relationship between the the visual prompt in the UI and the `examples.common.driver.ExampleDescriptor`. The `ExampleDescriptor` further defines, the order of the example in the list, the class string for example to be executed, any arguments (if any) and the class string for the controller that will guide the user experience when executing the example.

The primary controller of the main screen is located at `examples.common.driver.ExamplesController`. The `examples.common.driver.ExamplesView` is designed to realize the list of examples. The `ExamplesController` allows the user to select an example from the list and uses the `examples.common.driver.ExampleLauncher` to launch the example.

The `ExampleLauncher` can launch a controller or an example. If the `ExampleLauncher` is used to launch an example, the example is executed synchronously and after the example completes, control will return to the primary screen. If the `ExampleLauncher` is used to launch a controller, the controller is instantiated and executed. The new controller will need to execute the example and (optionally) provide a view for the UI. When the controller is completed, it will return the user back to the previous screen (usually the primary examples screen).

Compiling The Bundle Admin Client

Please find the build.xml file in the `OSGiexamples` directory of the CLDC environment to build the bundle admin client for the CLDC platform.

The `build.xml` uses the Netbeans Mobility Ant tasks to build bundle admin Midlet application.

The build.xml file uses a `properties.xml` as an interface to the developer's environment. Before attempting to use the `build.xml`, create a `properties.xml` file based on the contents of the `properties.xml.sample`.

The `README.txt` file also contains fine-grain details about the example and the build process. The following sections highlight several aspects of the build.

The developers build process must

- Unzip all core components from the `ve-core-cldc.jar` file.

  - This jar is required to build any Voyager application on the CLDC platform. It requires all the core Voyager components on the CLDC platform.

- Unzip the contents of the `ve-osgi-bundleAdminServer-client-public.jar` and compile the java interfaces.

  - This jar is required to build the bundle admin java client. The contents of this jar was generated when building the bundle admin server. It contains the public interfaces as well as the properties required to connect to the server instance.

- Preverify and compile all source code.

- `Pgen` the classes of all required public admin interfaces and implementations used by the bundle admin client.

- Build the Jad and Jar of the MIDLet application.

Please consult the `build.xml` for details about how this is accomplished.

The output of this build process produces a Jad/Jar pair that will reach out to the bundle admin server and allow the user to exercise any of the examples.

Running The Bundle Admin Client

The same `build.xml` file used to compile the Jad/Jar files for the bundle admin client also allows you to execute this application.

Before attempting to start the bundle admin client, ensure that the Bundle Admin Server has been successfully launched and the Voyager instance has been successfully initialized.

The examples illustrate how to use the `IOSGIAdminFacade`. These examples are coded to work with the `SimpleBundle` (bundled with the `bundleAdminServer`) so the bundle being administered is guaranteed to be available.

When the bundle admin client is launched the user is presented with many different examples. Below is a list of each example listed with a brief description.

- *List Bundles*

  - Lists all the bundles currently installed on the OSGi instance hosting the bundle admin server.

- *Install Simple Bundle, Uninstall Simple Bundle, Check Status of Simple Bundle, Start Simple Bundle, Stop Simple Bundle, Refresh Simple Bundle.*

  - These examples provide basic bundle administration to the `SimpleBundle`.

- o After the example completes, the application will notify the user about the results.

- o Note that the examples have been written to administer to the `SimpleBundle`, however, the `IOSGIAdminFacade` can be applied to any valid bundle.

- o Note, based on the OSGi specification, the *Refresh Simple Bundle* does not generate a bundle event on the OSGi instance. The Refresh action will generate a framework event.

- *Start Bundle Event Listener, Stop Bundle Event Listener, View Bundle Events*

  - o The OSGi framework will generate bundle events based the administration of any bundle on that instance.

  - o If the client starts the bundle event listener, this MIDLet application will also be notified, through Voyager, about any bundle events published by the OSGi instance (hosting the bundle admin server).

  - o The user can execute the basic bundle administration examples (except for *Refresh Simple Bundle*) when listening for bundle events in order to receive confirmation about the action performed (if successful).

  - o The user can *View Bundle Event* to view and/or clear the log of bundle events received about the remote OSGi framework instance.

- *Start Framework Listener, Stop Framework Listner, View Framework Events*

  - o The OSGi framework will generate framework events based on framework administration actions on that instance.

  - o If the client starts the framework event listener, this MIDLet application will also be notified, through Voyager, about any framework events published by the OSGi instance (hosting the bundle admin server).

  - o The user can execute the *Refresh Simple Bundle* example when listening for framework events in order to receive confirmation about the action performed (if successful).

  - o The user can *View Framework Events* to view and/or clear the log of framework events received about the remote OSGi instance.

- *Start Service Event Listener, Stop Service Event Listener, View Service Events*

  - o The OSGi framework will generate service events based on changes to services registered with the framework instance.

- If the client starts the service event listener, this MIDLet application will also be notified, through Voyager, about any service events published by the OSGi instance (hosting the bundle admin server).

- The user can execute *Register Yellow Pages Service* and *Unregister Yellow Pages Service* examples when listening for service events in order to receive confirmation about the action performed (if successful).

- The user can execute *View Service Events* to view and/or clear the log of service events received about the remote OSGi instance.

- *Register Yellow Pages, Unregister Yellow Pages*

  - These examples will register and unregister Voyager's Yellow Pages service in the OSGi Service Registry of the OSGi instance hosting the bundle admin server.

  - These examples do not directly use Voyager's Yellow Pages services. The examples are only provided to generate service events on the OSGi instance hosting the bundle admin server.

Yellow Pages Example

The Yellow Pages Example, which implements the same functionality as the example of the same name found in the `examples\java\se-cdc\java\examples\yellowpages` directory, consists of three instances of Voyager providing services and one instance using the services. Each of the three service instances runs as an OSGi bundle, and the client executes as a standalone Java application. The `README`.txt file describes the structure of the application.

Building the Yellow Pages Example

Successfully editing the `properties.xml.example` to produce an appropriate `properties.xml` requires knowing the following.

- The location of the OSGi framework's directory of jars/bundles, and the name of the OSGi framework startup jar.

- The prototype property file assumes that the `VOYAGER_HOME` environment variable is correctly set. If it is not, the single reference to the Voyager installation directory will need to be changed.

- The URLs for the three Voyager servers are defined, and contain default port numbers. If the default port numbers are already in use, the port numbers will need to be changed.

Once the `properties.xml` file has been created and edited the example source code can be compiled and the bundle jars assembled. Executing the `ant` command in the yellow pages example directory accomplishes this task, producing three bundles and some `.class` files.

Running the Yellow Pages Example

After successfully building the example, start the Voyager service bundles by starting a command shell, changing to the yellow pages example directory, and entering the command `ant start-stockmarket-example-services`. The Ant task will start the OSGi container and, based on the configuration file written during the build step, start the broker, PE, and trend bundles. The `readme.txt` file contains the output produced by a successful startup.

After all three bundles start successfully, in a second command shell start the example client, which executes as a stand-alone Java application, using `ant start-stockmarket-example-client`. The readme.txt file also contains the output produced by successful execution of the yellow pages client.

Normal termination of the OSGi framework is by entering `exit` in the command shell. The client program terminates by itself after displaying the buy/sell/hold recommendations received from the services.