

# Open Print Assist User Guide

## v0.1

WARNING: OPA is in early development. The current API is likely to change in future releases.

### Contents

About.....	2
Overview of OPA.....	2
Application Start-up.....	2
Example Module.....	3
API Summary.....	5
Appendix A – Example Module Code.....	8

## About

This guide is aimed at those developers wishing to use the OPA framework to develop new applications. Developers interested in contributing to the OPA project itself should review the Open Print Assist Developers Guide. This guide provides an overview of HOW TO CREATE USER SUPPLIED MODULES for the OPAKernel.

Please note that at this stage the OPAKernel API (the interface to the OPAKernels services) is constantly evolving and is subject to change.

## Overview of OPA

The OPA project is developing a Print Management System for instant print stores using Java. This is based upon an application framework that provides services common to most GUI applications. The term OPA refers to the framework itself.

This framework provides code useful to many applications so that developers have a head start. This saves valuable time and effort for new projects.

OPA is based on a modular architecture. The framework itself is generally used UNMODIFIED. That is, developers using the framework supply MODULES (for information on modifying the OPAKernel, see the OPA Developer Guide). These modules perform the business function of the application, the real work. The modules use the services provided by OPA such as window management (eg. creating dialog boxes), option management (eg. storing user preferences) and resource management (eg. providing access to localised text or icons common to GUI applications) and more.

By providing common services in a framework and using a modular architecture, application development can be easily distributed. Geographically isolated teams can work on separate modules with little need for interaction. To access the services that another module provides, a module only needs to know the interface that the other module implements.

When the interface is known a module can ask OPA to find a service (object instance – another module) implementing the interface, or even create a new instance if one doesn't exist.

## Application Start-up

This section covers an OPA based application start-up. Note that modules are loaded into OPA in a *push* fashion. That is, modules are responsible for registering or activating the services they need. This is done with simple Java statements. These statements are called by the OPAKernel's module loader at start-up.

When an OPA based application starts, the following happens:

- Java Virtual Machine (JVM) executes OPAKernel class
- OPAKernel executes `run()` method
- OPA finds a module list which includes the registered modules and passes the list to the OPAModuleLoader.
- The OPAModuleLoader expects a JAR file with the same name as the modules in the module list. It mounts these JAR files then calls the Loader class in the JAR file. Each OPA Module

must have a class that implements the Loader interface. The Loader interface defines a `load()` method which is the list of commands the module needs to load itself into the OPA framework.

- The `load()` method statements are executed and the modules register help information, service names, request toolbars and menus etc.
- After the modules are loaded the OPAKernel starts the framework services such as the OPAHelpManager.
- Then OPA builds the toolbars, menus and OPADesktop and opens the application.

### Example Module

This section explains HOW TO CREATE A MODULE and provides a concrete example.

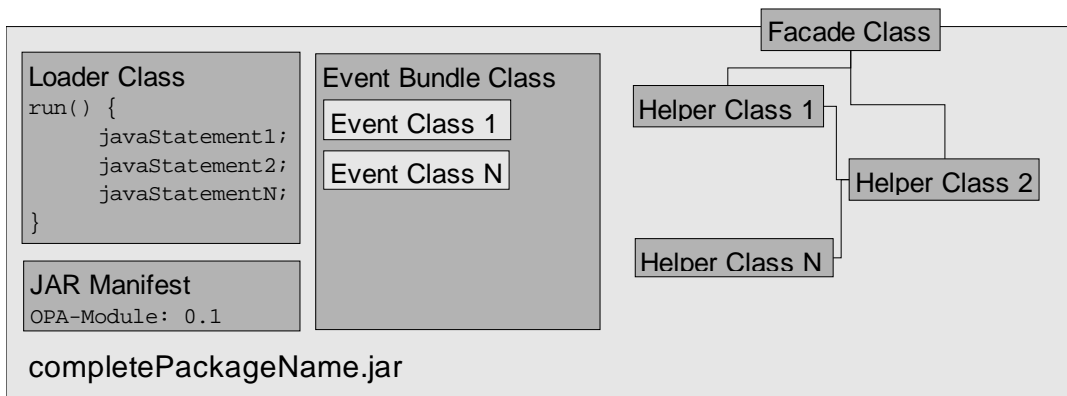
NOTE: An OPA Module must be named as follows:

*completePackageName.jar*

Eg. If the package is `moduleInstaller` in the OPA project, then the OPA Module name must be *org.opa.moduleInstaller.jar*. In the JAR manifest there MUST be an entry `OPA-Module: 0.1`. Where *OPA-Module* is the key and *0.1* is the value. This is used by the `OPAModuleLoader` at startup.

The Loader class should add the events to the event bundle, load the event bundle into the `OPADisplayBuilder`, register the facade class as a service with the `OPAServiceManager` and do other tasks the module needs. This code is added by the user developer.

The facade class should be the only one visible to other modules. It should invoke other 'helper' classes.



The JAR Manifest MUST include an entry describing the JAR as an OPA Module.

The OPA Module must be named after the package structure inside the module.

Figure 1 - OPA Module Architecture

There are five (5) classes needed for an OPAModule:

1. `XXXEvent.class` – this class MUST extend `org.opa.kernel.OPAAbstractEvent`. This class describes an event and what menus and/or toolbars are needed. The `actionPerformed()` method

provides the code to handle the event. usesToolbar() and usesMenu() return information to OPA about if this event needs a menu item (application's menubar) or toolbar (a separate toolbar for each bundle, separate button for each event). getMenuPath() returns a "/" delimited string that specifies the menu structure. getIcon() and getLabel() return an icon and text string respectively to be used for the toolbar (icon only) button and menu button.

2. XXXBundle.class – this class must EXTEND org.opa.kernel.OPAAbstractEventBundle. The name of the class is not important. It MUST override the load() method with statements of the form add(new myEventName()); where myEventName is the name of an event class (see above).
3. Loader.class – this class MUST implement org.opa.kernel.Loader AND MUST be called Loader.class (this may change in future releases). The load() method is overridden and commands are added that the module uses to load into the OPAKernel. An important command is to add the XXXBundle.class to the OPADisplayBuilder in OPA. If this is not added no menus or toolbars will be created.
4. A facade class for the module – this class is the entry point to the modules services. It is best to have ONLY ONE (1) facade class. This makes code more secure. This facade class should be registered with the OPAKernel (by Loader.class). After it is registered with the OPAKernel, other modules can use the OPAKernel to find the service.

See Appendix A for a complete copy of the above classes in an actual OPA Module.

## API Summary

This section summarises the OPAKernel interface.

**OPADisplayBuilder** – Handles the creation of application menus and toolbars.

```
/** Adds an OPAEventBundle to be used in building the toolbars and menubar. */
void add(OPAEventBundle bundle);

/** Builds and returns the toolbar panel for the main frame. */
OPAToolBar buildToolBar();

/** Builds and returns the OPAMenubar for the main frame */
JMenuBar buildMenuBar();
```

**OPAHelpManager** – Provides simple access to help information

```
/** Registers a help location with the HelpManager. This
 * is implementation dependent. Typically the key is some
 * unique code and the target is an external system command,
 * URL or OPAPanel.
 */
void registerHelp(Object key, Object target);

/** Requests help from the help manager. This is
 * implementation dependent. How this is handled is
 * upto the individual HelpManagers. They should however
 * provide some default mechanism if the request cannot
 * be found.
 */
void getHelp(Object key);
```

**OPAOptionManager** – handles user preferences

```
/** Adds a default Properties object to the system option
 * manager. This is called by each modules 'Loader' object
 * at boot time. These properties are static and options
 * selected by each user are layed over these to create
 * the in-use system options.
 */
void addOption(Properties moduleDefaults);

/** retrieves an options value. The key must begin with the module
 * name to avoid naming conflicts within the entire application.
 */
String get(String key);

/** Sets an options value. If the 'key' is invalid, the call is
 * disregarded and has no effect.
 */
void set(String key, String value);

/** Starts the OPAOptionManager so that it can begin handling
 * option changes.
 */
void start();

/** Stops the OPAOptionManager and tidys up its in memory data
```

```

* structures
*/
void stop();

```

### OPAResourceManager – provides text strings and images common to most GUI applications

```

/** Returns a resource to a client, typically an OPA module.
* The client can request resources such as localised text or
* icon images by providing a string key. Implementations should
* make the keys available in the public interface as static final
* Strings.
* Clients cannot register resources, but only make use of the
* ones provided by the platform. Module specific resources can
* be provided internally by the module itself using the getResource()
* method of Class or by subclassing ListResourceBundle.
*/
public Object getResource(String key);

```

### OPAServiceManager – locates and/or starts service (facade class) instances

```

/** Registers a service with the service manager. This can then
* be queried later by clients requesting a particular service.
* Services are registered at boot time when modules are loaded.
* The 'Loader' class provided by each module is responsible for
* 'pushing' the registration into the kernel.
* Services are indexed by the interfaces they implement and there
* can only be a SINGLE implementation of each interface registered.
* The facade class that is registered should be the public interface
* to a package (OPAModule). Events supplied by the module itself may
* access other module members directly, by name without having to
* use the ServiceManager.
*/
void registerService(Class reference, Object serviceFacade);

/** Returns the facade class registered for the Interface provided in
* the query. Note that the returned Object may implement any number of
* other interfaces.
*/
Object locateService(Class reference);

/** Returns an array of registered facade classes. It contain the direct
* references to the classes.
*/
Object[] listServices();

/** Returns an array of the interfaces now registered in the service manager.
* To obtain the facade class implementing the interface, pass the return
result
* to the 'locateService()' method.
* Note the interfaces returned are ONLY THOSE USED IN THE SERVICE INDEX.
Other
* interfaces may be implemented but not returned on this method call.
*/
Class[] listInterfaces();

```

### OPAWindowManager – creates dialog boxes, finds and/or creates panels and adds them to dialogs

```

/** Returns the specified dialog type loaded with the panel
* specified by name. The name must match the name of the

```

```
    * panel class file otherwise an error is returned. */
OPADialog dialogByName(String name);

/** Returns the specified internal frame type loaded with the panel
 * specified by name. The name must match the name of the
 * panel class file otherwise an error is returned. */
OPAInternalDialog internalByName(String name);

/** Returns the MasterOptionPanel. This should only be called by the Loader
classes
 * when adding sub-panels at boot time.
 */
OPAMasterOptionPanel getMaster();
```

## Appendix A – Example Module Code

Copyright (c) 2003, Open Print Assist (openprintassist.sourceforge.net)  
All rights reserved.

Original design and code Copyright (c) 2003, Michael Dawson  
All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notices, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notices, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the names of MICHAEL DAWSON, OPEN PRINT ASSIST (OPA) nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

## ModuleInstallerEvent.java

```
package org.opa.moduleInstaller;

import org.opa.kernel.*;
import java.util.*;
import javax.swing.*;

public class ModuleInstallerEvent extends org.opa.kernel.OPAAbstractEvent {

    /** Creates a new instance of OptionEvent */
    public ModuleInstallerEvent() {
    }

    public void actionPerformed(java.awt.event.ActionEvent e) {
        org.opa.kernel.OPAWindowManager man = kernel.getWindowManager();
        OPAInternalDialog dialog = man.internalByName
            ("org.opa.moduleInstaller.OPADefaultModuleInstallerPanel");
        dialog.getHelper().show(null, null);
        kernel.getLogManager().viewableLog("Module installer opened...");
    }

    public ImageIcon getIcon() {
        return (ImageIcon) kernel.getResourceManager().getResource
            ("ModuleIconSmall");
    }

    public String getLabel() { // just for toolbar buttons
        return null; //(String) ResourceBundle.getBundle
            ("unneeded.moduleInstaller.ModInstResource").getObject("OKButton");
    }

    public String getMenuPath() {
        return "Tools/Modules/Module Loader";
    }

    public boolean usesToolBar() {
        return true;
    }

    public boolean usesMenu() {
        return true;
    }
}
```

## **ModuleInstallerBundle.java**

```
package org.opa.moduleInstaller;

public class ModuleInstallerBundle extends org.opa.kernel.OPAAbstractEventBundle
{
    /** Creates a new instance of TestBundle */
    public ModuleInstallerBundle() {
    }

    public void load() {
        add(new ModuleInstallerEvent());
    }
}
```

## Loader.java

```
package org.opa.moduleInstaller;
import org.opa.kernel.*;
import java.util.*;
import java.net.*;

public class Loader implements org.opa.kernel.OPALoader {

    /** Creates a new instance of Loader */
    public Loader() {
    }

    public void load() {
        kernel.getDisplayBuilder().add(new ModuleInstallerBundle());
        try {
            kernel.getServiceManager().registerService(Class.forName(
                "org.opa.kernel.OPAModuleInstaller"),
                new OPADefaultModuleInstaller(kernel));
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public void setKernel(OPAKernel kernel) {
        if(kernel != null)
            this.kernel = kernel;
    }

    private OPAKernel kernel;
}

```

## **OPADefaultModuleInstaller.java – this is the *facade* class for this module**

```
package org.opa.moduleInstaller;

import org.opa.kernel.*;

/**
 *
 * @author md
 */
public class OPADefaultModuleInstaller implements OPAModuleInstaller {

    private OPAKernel kernel;

    /** Creates a new instance of OPADefaultModuleInstaller */
    public OPADefaultModuleInstaller(OPAKernel kernel) {
        if(kernel != null)
            this.kernel = kernel;
    }

    public void start() {
        kernel.getWindowManager().internalByName
            ("org.opa.moduleInstaller.OPADefaultModuleInstallerPanel").getHelper
            ().show(null, null);
    }
}
```

## OPADefaultModuleInstallerPanel.java – helper class for this module

```
package org.opa.moduleInstaller;

import org.opa.kernel.*;
import java.io.File;
import javax.swing.*;
import javax.swing.filechooser.*;

public class OPADefaultModuleInstallerPanel extends OPAPanel {

    /** Creates new form OPADefaultModuleInstallerPanel */
    public OPADefaultModuleInstallerPanel() {
        initComponents();
    }

    /** This method is called from within the constructor to
     * initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is
     * always regenerated by the Form Editor.
     */
    private void initComponents() { //GEN-BEGIN: initComponents
        java.awt.GridBagConstraints gridBagConstraints;

        summaryField = new javax.swing.JTextField();
        locateField = new javax.swing.JTextField();
        installField = new javax.swing.JTextField();
        finishField = new javax.swing.JTextField();

        setLayout(new java.awt.GridBagLayout());

        summaryField.setBackground((java.awt.Color)
            javax.swing.UIManager.getDefaults().get("windowBorder"));
        summaryField.setEditable(false);
        summaryField.setText("This will install a new module");
        summaryField.setBorder(null);
        summaryField.setFocusable(false);
        gridBagConstraints = new java.awt.GridBagConstraints();
        gridBagConstraints.gridx = 0;
        gridBagConstraints.gridy = 0;
        gridBagConstraints.anchor = java.awt.GridBagConstraints.WEST;
        gridBagConstraints.insets = new java.awt.Insets(4, 14, 4, 4);
        add(summaryField, gridBagConstraints);

        locateField.setBackground((java.awt.Color)
            javax.swing.UIManager.getDefaults().get("windowBorder"));
        locateField.setEditable(false);
        locateField.setFont(new java.awt.Font("Dialog", 1, 12));
        locateField.setText("1. Locate module");
        locateField.setBorder(null);
        locateField.setDisabledTextColor(new java.awt.Color(0, 0, 0));
        locateField.setFocusable(false);
        locateField.setMinimumSize(new java.awt.Dimension(300, 16));
        locateField.setPreferredSize(new java.awt.Dimension(300, 16));
        gridBagConstraints = new java.awt.GridBagConstraints();
        gridBagConstraints.gridx = 0;
        gridBagConstraints.gridy = 1;
        gridBagConstraints.anchor = java.awt.GridBagConstraints.WEST;
        gridBagConstraints.insets = new java.awt.Insets(4, 14, 4, 4);
        add(locateField, gridBagConstraints);
    }
}
```

```

installField.setBackground((java.awt.Color)
    javax.swing.UIManager.getDefaults().get("windowBorder"));
installField.setEditable(false);
installField.setText("2. Install module");
installField.setBorder(null);
installField.setFocusable(false);
installField.setMinimumSize(new java.awt.Dimension(300, 16));
installField.setPreferredSize(new java.awt.Dimension(300, 16));
gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 0;
gridBagConstraints.gridy = 2;
gridBagConstraints.anchor = java.awt.GridBagConstraints.WEST;
gridBagConstraints.insets = new java.awt.Insets(4, 14, 4, 4);
add(installField, gridBagConstraints);

finishField.setBackground((java.awt.Color)
    javax.swing.UIManager.getDefaults().get("windowBorder"));
finishField.setEditable(false);
finishField.setText("3. Finish");
finishField.setBorder(null);
finishField.setFocusable(false);
finishField.setMinimumSize(new java.awt.Dimension(70, 16));
finishField.setPreferredSize(new java.awt.Dimension(70, 16));
gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 0;
gridBagConstraints.gridy = 3;
gridBagConstraints.anchor = java.awt.GridBagConstraints.WEST;
gridBagConstraints.insets = new java.awt.Insets(4, 14, 4, 4);
add(finishField, gridBagConstraints);

} //GEN-END:initComponents

public void button1Action() {
    phase++;
    if(phase > 3)
        window.destroy();
    adjust();
}

public void button2Action() {
    if(phase == 2) {
        phase = 0;
        locateField.setText("1. Locate module");
    }
    else
        phase--;
    if(phase < 0)
        window.destroy();
    adjust();
}

public void button3Action() {
}

public String getPanelName() {
    return "Module Installer";
}

public void button4Action() {

```

```

}

public void onCreate() {
}

public void onDestroy() {
}

public void onLoad() {
}

public void onLoad(Object obj1, Object obj2) {
    phase = 0;
}

public void onUnload() {
}

private void adjust() {
    switch(phase) {
        case 0:
            locateField.setFont(new java.awt.Font("Dialog", 1, 12));
            installField.setFont(new java.awt.Font("Dialog", 0, 12));
            finishField.setFont(new java.awt.Font("Dialog", 0, 12));
            break;
        case 1:
            if(fc == null) {
                fc = new JFileChooser();
                fc.addChoosableFileFilter(new ModuleFilter());
            }
            int returnVal = fc.showOpenDialog(this);
            if(returnVal == JFileChooser.APPROVE_OPTION) {
                file = fc.getSelectedFile();
                locateField.setText("1. Locate module" + ": " +
                    file.getAbsolutePath());
                phase++;
                adjust();
            }
            else {
                phase--;
                adjust();
            }
            break;
        case 2:
            locateField.setFont(new java.awt.Font("Dialog", 0, 12));
            installField.setFont(new java.awt.Font("Dialog", 1, 12));
            finishField.setFont(new java.awt.Font("Dialog", 0, 12));
            break;
        case 3:
            try {
                new OPAModuleVerifier().install(file);
            } catch(Exception e) {
                System.out.println(e.getMessage());
                e.printStackTrace();
                phase--;
                break;
            }
            locateField.setFont(new java.awt.Font("Dialog", 0, 12));
            installField.setFont(new java.awt.Font("Dialog", 0, 12));
            installField.setText(installField.getText() + ":
                Successful!");
            finishField.setFont(new java.awt.Font("Dialog", 1, 12));
            break;
    }
}
}

```

```

public void button5Action() {
}

public Object[] getControls() {
    return new Object[] {
        (String)kernel.getResourceManager().getResource("ForwardText"), (Icon)
            kernel.getResourceManager().getResource
                (OPADefaultResourceManager.FORWARD_ICON_LARGE),
        (String)kernel.getResourceManager().getResource("BackText"),
        (Icon)kernel.getResourceManager().getResource
            (OPADefaultResourceManager.BACK_ICON_LARGE)
    };
}

public void OKAction() {
}

class ModuleFilter extends javax.swing.filechooser.FileFilter {

    public boolean accept(File f) {
        if(f.isDirectory())
            return true;
        String extension = Utils.getExtension(f);
        if(extension != null) {
            if(extension.equals(Utils.jar))
                return true;
            else
                return false;
        }
        return false;
    }

    public String getDescription() {
        return "OPA modules";
    }
}

private javax.swing.JTextField finishField;
private javax.swing.JTextField installField;
private javax.swing.JTextField locateField;
private javax.swing.JTextField summaryField;
private int phase = 0;
private JFileChooser fc;
private File file;
}

class Utils { // created by Sun Microsystems www.java.sun.com

    public final static String jpeg = "jpeg";
    public final static String jpg = "jpg";
    public final static String gif = "gif";
    public final static String tiff = "tiff";
    public final static String tif = "tif";
    public final static String png = "png";
    public final static String jar = "jar";

    /*
     * Get the extension of a file.
     */
}

```

```
public static String getExtension(File f) {
    String ext = null;
    String s = f.getName();
    int i = s.lastIndexOf('.');

    if (i > 0 && i < s.length() - 1) {
        ext = s.substring(i+1).toLowerCase();
    }
    return ext;
}
}
```

## OPAModuleVerifier.java – helper class for this module

```
package org.opa.moduleInstaller;

import java.util.jar.*;
import java.io.*;
import java.util.*;

/**
 *
 * @author md
 */
public class OPAModuleVerifier {

    private JarFile jar;

    /** Creates a new instance of OPAModuleVerifier */
    public OPAModuleVerifier() {
    }

    public void install(File candidate) throws Exception{
        jar = new JarFile(candidate);
        Attributes att = jar.getManifest().getMainAttributes();
        if(att.containsKey("0.1")) {
            if(att.getValue("OPA-Module").equals("0.1")) {
                File destination = new File(System.getProperty("user.home") +
                    "/" + candidate.getName());
                destination.createNewFile();
                BufferedOutputStream out = new BufferedOutputStream(new
                    FileOutputStream(destination));
                BufferedInputStream in = new BufferedInputStream(new
                    FileInputStream(candidate));
                int bite;
                while((bite = in.read()) >= 0) {
                    out.write(bite);
                }
                out.flush();
                in.close();
                out.close();
                Properties moduleList = new Properties();
                moduleList.load(new FileInputStream(System.getProperty
                    ("user.home") + "/.moduleList"));
                moduleList.setProperty("module" + moduleList.size(),
                    destination.getName().substring(0, destination.getName
                    ().length() - 4));
                moduleList.store(new FileOutputStream(System.getProperty
                    ("user.home") + "/.moduleList"), null);
            }
        }
        else {
            int nothing = 1;
        }
    }
}
```