Data Crow
Developer Guide

Tips and tricks for developers

Written by Robert Jan van der Waals | June 16, 2009 |
Version 0.1 | Based on Data Crow 3.4.9

Introduction

This guide is meant to support developers in creating new plugins and online services. It strives to provide the reader (you) with a little bit more insight in how Data Crow works.

Note that it is a very good idea to use the latest version of the *Javadoc* which describes the code structure as well as some additional comments on the mentioned methods and classes. You can find this on http://www.datacrow.net.

The UML models copied into this document stem from a large UML diagram in a *zargo* file format (I have used ArgoUML to create the diagrams).

It would be great if you actually wrote additional code such as a plugin! It would even be better for you to share the new plugin with the rest of the Data Crow community. Either post a message on one of the forums or email me on info@datacrow.net.

Contents

1 Working with items

Each Data Crow item is represented by the *DcObject* class (or one of its many sub classes). This class allows items to be saved, updated, created and deleted.

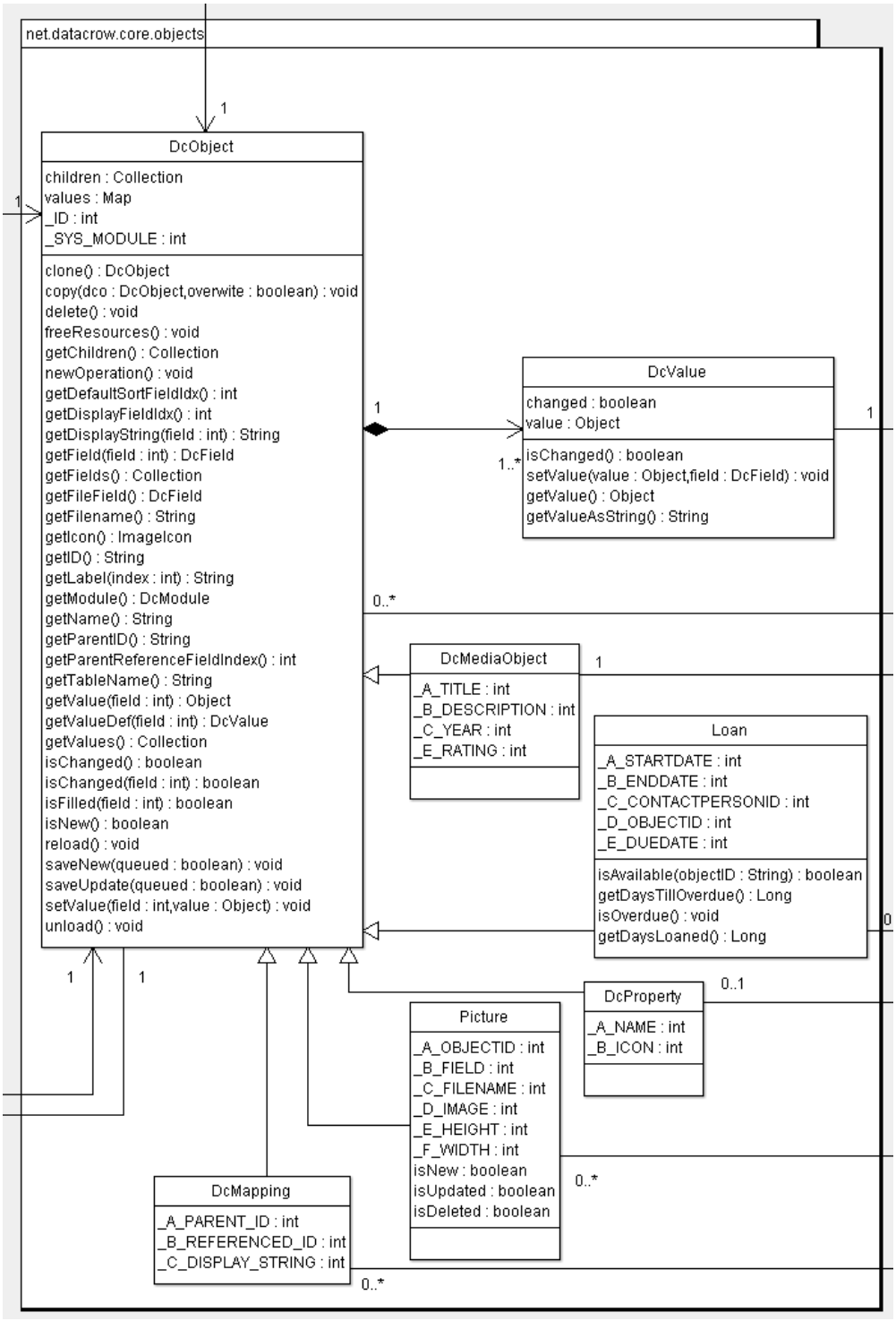
Each item is represented by a module (modules are explained in the next chapter). These modules hold the defined fields belonging to the item.

Both modules and item definitions are generic within Data Crow. This has great advantages for users but makes development a bit harder. The first thing you might notice is that none of the Data Crow object item classes (such as the *Software* and the *Movie* class) have specific getter and setter methods for their properties. Instead the properties are stored in a map. Fields are index based. These indices are defined in the XML definition of the module (see the chapter about modules for more information on this). I have, for all standard modules, written so called helper classes. These have simple class field defined holding the name and index for each of the fields, like so:

```
public static final int _F_DEVELOPER = 1;
public static final int _G_PUBLISHER  = 2;
public static final int _H_PLATFORM = 3;
public static final int _I_WEBPAGE = 4;
public static final int _K_TYPE = 6;
public static final int _L_MULTI = 7;
```

In case you want to do some additional development work on one of your custom modules I would suggest doing the same thing. If you prefer not doing this you will have to remember the indices of the fields or use their system name (as defined in the XML structure).

1.1 UML Model


net.datacrow.core.objects
DcObject
children : Collection
values : Map
_ID : int
_SYS_MODULE : int
clone() : DcObject
copy(dco : DcObject,overwite : boolean) : void
delete() : void
freeResources() : void
getChildren() : Collection
newOperation() : void
getDefaultSortFieldIdx() : int
getDisplayFieldIdx() : int
getDisplayString(field : int) : String
getField(field : int) : DcField
getFields() : Collection
getFileField() : DcField
getFilename() : String
getIcon() : ImageIcon
getID() : String
getLabel(index : int) : String
getModule() : DcModule
getName() : String
getParentID() : String
getParentReferenceFieldIndex() : int
getTableName() : String
getValue(field : int) : Object
getValueDef(field : int) : DcValue
getValues() : Collection
isChanged() : boolean
isChanged(field : int) : boolean
isFilled(field : int) : boolean
isNew() : boolean
reload() : void
saveNew(queued : boolean) : void
saveUpdate(queued : boolean) : void
setValue(field : int,value : Object) : void
unload() : void
DcValue
changed : boolean
value : Object
isChanged() : boolean
setValue(value : Object,field : DcField) : void
getValue() : Object
getValueAsString() : String
DcMediaObject
_A_TITLE : int
_B_DESCRIPTION : int
_C_YEAR : int
_E_RATING : int
Loan
_A_STARTDATE : int
_B_ENDDATE : int
_C_CONTACTPERSONID : int
_D_OBJECTID : int
_E_DUEDATE : int
isAvailable(objectID : String) : boolean
getDaysTillOverdue() : Long
isOverdue() : void
getDaysLoaned() : Long
DcProperty
_A_NAME : int
_B_ICON : int
Picture
_A_OBJECTID : int
_B_FIELD : int
_C_FILENAME : int
_D_IMAGE : int
_E_HEIGHT : int
_F_WIDTH : int
isNew : boolean
isUpdated : boolean
isDeleted : boolean
DcMapping
_A_PARENT_ID : int
_B_REFERENCED_ID : int
_C_DISPLAY_STRING : int
1
1..*
0..*
0..1

1.2 Getting an item

An item can be created from a module by calling the *getDcObject* method. This creates a new item which is not yet represented in the database.

```
DcModules.get("software").getDcObject();
```

To find an existing item you will have to use the *DataManager*. The DataManager is the persistency layer of Data Crow. The next chapter explains how to use this persistency layer.

1.3 Working with values

The DcObject class does not have specific getters and setters for its values but uses (because of Data Crow's flexible nature) a map to store and reference its values.

```
DcObject dco = DcModules.get("software").getDcObject();
// to set a value
dco.setValue(7, "Test");
// to get a value
String test = (String) dco.getValue(7);
```

As you can see everything is referenced via indices (field 7 in the example above). But how do you know which index is used for which field? This is explained in the module XML chapter.

1.4 Saving an Item

An item can be saved by using the *saveNew* or the *saveUpdate* methods. The *saveUpdate* method is used to save changes for existing items. The *saveNew* method should be used to store new items. By supplying the *queued* boolean you can indicate if the item should be saved using the *QueryQueue* or if it should be saved directly from within the current thread. Using he QueryQueue (see javadoc) is the preferred method of saving but has the downside of not knowing when the save action has been finished.

1.5 Deleting an Item

An item can be deleted by using the *delete* method.

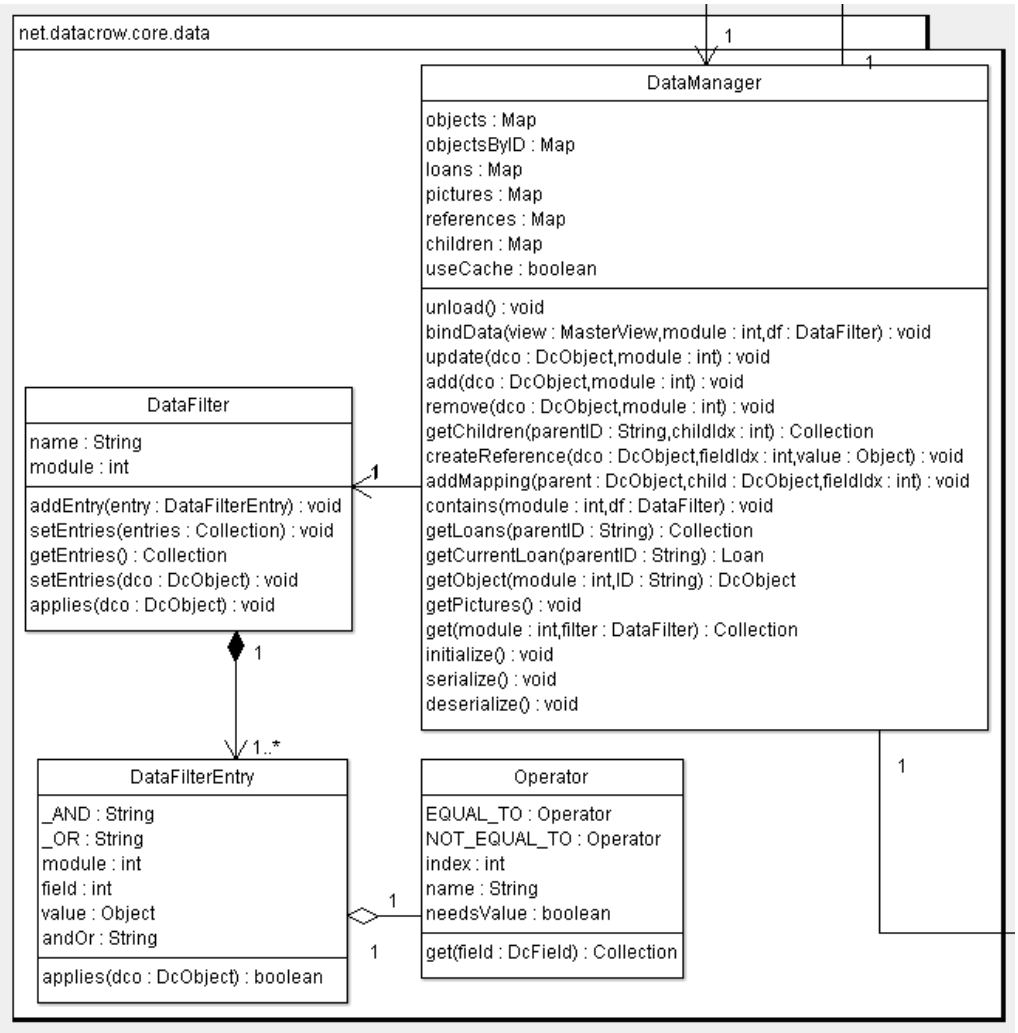
2 The Persistency Layer

The persistency layer if fully represented by the *DataManager* class (*net.datacrow.core.data.DataManager*). This class is responsible for loading, retrieving, updating, creating and maintaining item information.

Most of the methods are used internally. You will not need to add or remove items directly to the persistency layer, this happens automatically when using the correct methods (such as the *saveUpdate* and *saveNew* methods of a *DcObject* item, see previous chapter).

A common function of the DataManager class is to retrieve items. This will be explained in this chapter.

2.1 UML Model



2.2 Retrieving an item by its id

The simplest way of retrieving an item is by searching on its ID.

```
DcObject dco = DataManager.getObject(DcModules._SOFTWARE, “1”)
```

In the example above the software item with ID one is retrieved.

2.3 Filtering

A data filter consists of entries. Each entry represents a condition to which an item must apply. A single entry is always based on a specific field of an item, an operator and, if applicable, a value. The value can be the query or condition to which the actual value of the field will be compared. The value is however not needed when using operators like ‘*is empty*’.

Below you find an example. The example searches for software items which do not have a value set for the *serial key* field.

```
DataFilter df = new DataFilter(DcModules._SOFTWARE);

df.addEntry(new DataFilterEntry(DataFilterEntry._AND, DcModules._SOFTWARE,
SOFTWARE._S_SERIALKEY, Operator.IS_EMPTY, null));

DataManager.get(DcModules._SOFTWARE, df);
```

You might have noticed that the module is defined for both the DataFilter as well as for the DataFilterEntry. The reason is that is possible to filter on parent items using filter entries based on their children. For example:

```
DataFilter df = new DataFilter(DcModules._AUDIOCD);

df.addEntry(new DataFilterEntry(DataFilterEntry._AND, DcModules._AUDIOTRACK,
AUDIOTRACK._A_TITLE, Operator.EQUALS, ”Track 1”));

DataManager.get(DcModules. AUDIOCD, df);
```

In the example above all audio CD’s are retrieved having one or more tracks with title “Track 1”.

3 Working with modules

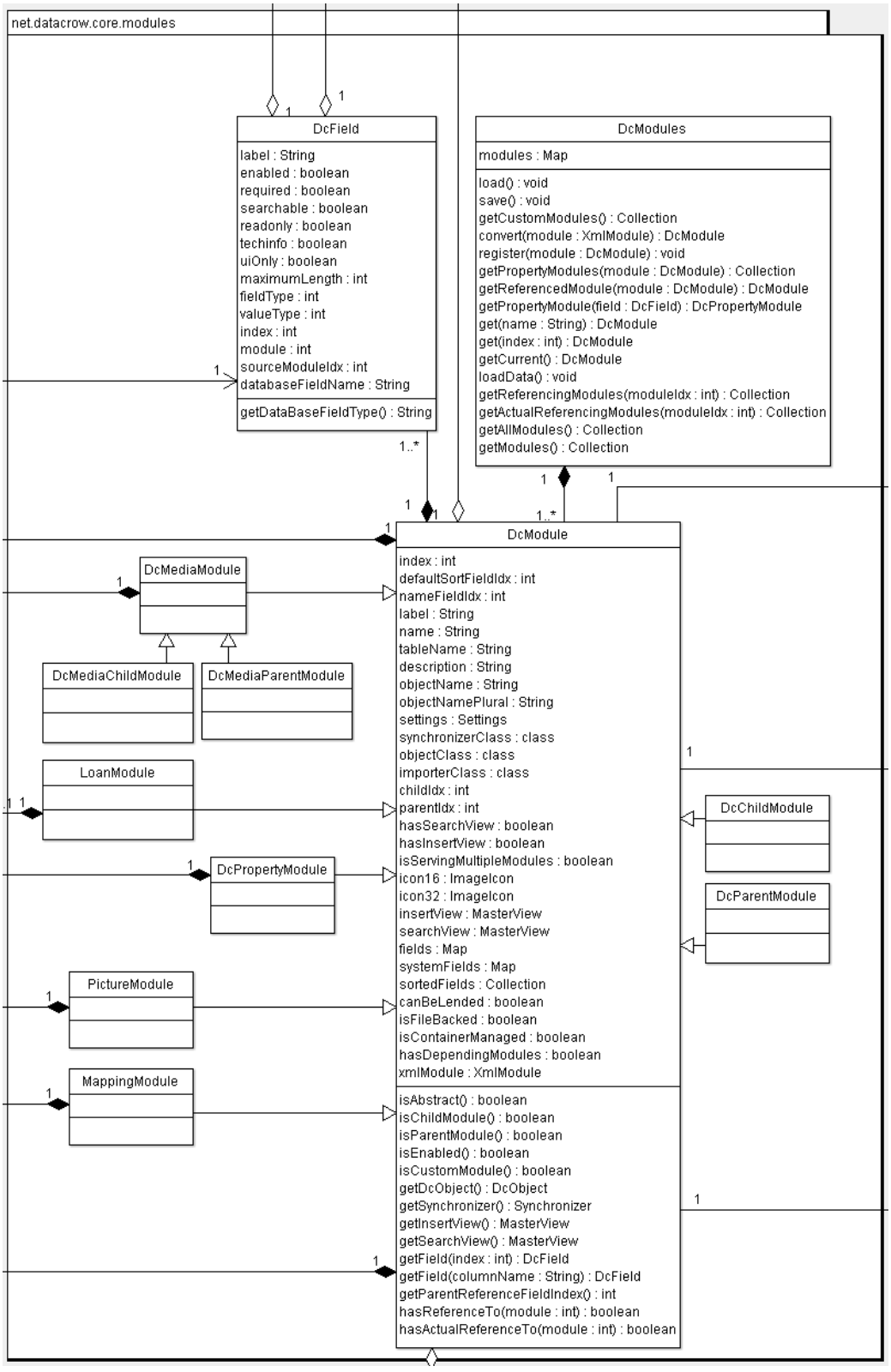
A module ties all the functionality together and maintains items. Each item within Data Crow belongs to a specific module. Using a module a new item, belonging to this module, can be created. Furthermore tools and utilities can be requested from module (such as the File Import utility).

Modules are stored in the *modules* directory of Data Crow, the so called module JAR files. Each jar file contains at least an XML definition of the module (module.xml). The XML definition of a module is explained in one of the next chapters.

Modules are registered in the *DcModules* class on startup of the application. This is done automatically. Modules can be retrieved from this class by their name or by using their index.

The database model belonging to the module is **fully** maintained by the Data Crow engine. It is created when not present and upgraded when needed. This means that you as a developer do not have to concern yourself with the database structure being used by your modules.

3.1 UML Model


net.datacrow.core.modules
DcField
label : String
enabled : boolean
required : boolean
searchable : boolean
readonly : boolean
techinfo : boolean
uiOnly : boolean
maximumLength : int
fieldType : int
valueType : int
index : int
module : int
sourceModuleIdx : int
databaseFieldName : String
getDataBaseFieldType() : String
DcModules
modules : Map
load() : void
save() : void
getCustomModules() : Collection
convert(module : XmlModule) : DcModule
register(module : DcModule) : void
getPropertyModules(module : DcModule) : Collection
getReferencedModule(module : DcModule) : DcModule
getPropertyModule(field : DcField) : DcPropertyModule
get(name : String) : DcModule
get(index : int) : DcModule
getCurrent() : DcModule
loadData() : void
getReferencingModules(moduleIdx : int) : Collection
getActualReferencingModules(moduleIdx : int) : Collection
getAllModules() : Collection
getModules() : Collection
DcModule
index : int
defaultSortFieldIdx : int
nameFieldIdx : int
label : String
name : String
tableName : String
description : String
objectName : String
objectNamePlural : String
settings : Settings
synchronizerClass : class
objectClass : class
importerClass : class
childIdx : int
parentIdx : int
hasSearchView : boolean
hasInsertView : boolean
isServingMultipleModules : boolean
icon16 : ImageIcon
icon32 : ImageIcon
insertView : MasterView
searchView : MasterView
fields : Map
systemFields : Map
sortedFields : Collection
canBeLended : boolean
isFileBacked : boolean
isContainerManaged : boolean
hasDependingModules : boolean
xmlModule : XmlModule
isAbstract() : boolean
isChildModule() : boolean
isParentModule() : boolean
isEnabled() : boolean
isCustomModule() : boolean
getDcObject() : DcObject
getSynchronizer() : Synchronizer
getInsertView() : MasterView
getSearchView() : MasterView
getField(index : int) : DcField
getField(columnName : String) : DcField
getParentReferenceFieldIndex() : int
hasReferenceTo(module : int) : boolean
hasActualReferenceTo(module : int) : boolean
DcMediaModule
DcMediaChildModule
DcMediaParentModule
LoanModule
DcPropertyModule
PictureModule
MappingModule
DcChildModule
DcParentModule

3.2 Retrieving a module by name

Each module has a unique name and this name can be used to retrieve the module.

```
DcModule modStamps = DcModules.get("software");
```

3.3 Retrieving a module by index

Every module gets a unique index assigned when created using the module creation wizard of Data Crow. This number is stored in the module XML (which is explained in the next chapter).

```
DcModule modStamps = DcModules.get(12);
```

3.4 Module XML structure

Modules are stored in an XML format and can be found in the modules directory of Data Crow. The XML file, containing the module definition, is stored in a JAR file. Below an example is shown of a module definition.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<modules xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="file://module.xsd">
  <module>
    <index>54</index>
    <product-version>3.4.5</product-version>
    <display-index>4</display-index>
    <label>Books</label>
    <name>book</name>
    <description>The book module contains all kinds of books. Here you can maintain and
search for books.</description>
    <key-stroke>null</key-stroke>
    <enabled>true</enabled>
    <can-be-lended>true</can-be-lended>
    <icon-16>icon16.png</icon-16>
    <icon-32>icon32.png</icon-32>
    <object-name>Book</object-name>
    <object-class>net.datacrow.core.objects.helpers.Book</object-class>
    <object-name-plural>Books</object-name-plural>
    <table-name>book</table-name>
    <table-name-short></table-name-short>
    <module-class>net.datacrow.core.modules.DcMediaModule</module-class>
    <child-module></child-module>
    <parent-module></parent-module>
    <has-search-view>true</has-search-view>
    <has-insert-view>true</has-insert-view>
    <is-file-backed>true</is-file-backed>
    <is-container-managed>true</is-container-managed>
    <importer-class>net.datacrow.fileimporters.EbookImport</importer-class>
    <synchronizer-class>net.datacrow.synchronizers.BookSynchronizer</synchronizer-class>
    <has-depending-modules>false</has-depending-modules>
    <default-sort-field-index>0</default-sort-field-index>
    <name-field-index>150</name-field-index>
    <is-serving-multiple-modules>false</is-serving-multiple-modules>
    <fields>
      <field>
        <index>1</index>
        <name>Publishers</name>
        <database-column-name>Publishers</database-column-name>
        <ui-only>true</ui-only>
        <enabled>true</enabled>
```

```
            <readonly>false</readonly>
            <searchable>true</searchable>
            <techinfo>false</techinfo>
            <maximum-length>20</maximum-length>
            <field-type>21</field-type>
            <module-reference>33000</module-reference>
            <value-type>18</value-type>
            <overwritable>false</overwritable>
          </field>
        </fields>
    </module>
</modules>
```

3.5 Modules

Here follows an explanation on each of the properties mentioned above. The bold properties are the ones that are the most interesting, or basic, properties.

Property	Description
Index	The unique module index.
Product-version	The product version to which this module belongs. This field is managed by the Data Crow.
Display-index	Decides the position of this module in the module bar.
Label	The display label.
Name	The unique name of the module.
Description	A short description of the module. This description is displayed as a popup in the module bar.
Key-stroke	The key combination to select the module.
Enabled	Indicates if this module is enabled.
Can-be-lended	Activates the loan management functionality when enabled.
Icon-16	The small icon filename (icon16.png by default).
Icon-32	The large icon filename (icon32.png by default).
Object-name	The name of the items managed by this module.
Object-class	The class used to represent items managed by this module. The defined class should extend the *DcObject* class. Using specialized object classes will help you when developing additional features (such as plugins) for your module.
Object-name-plural	The plural name for items managed by this module.
Table-name	The database table name.
Table-name-short	The database table short name.
Module-class	The class used to represent this module. This can either be the dynamic *DcModule* class or, when needed, a specialized implementation extending the *DcModule* class.
Child-module	The index of the child module (if any).
Parent-module	This index of the parent module (if any).
Has-search-view	Indicates this module has a search view available.
Has-insert-view	Indicates this module has an insert view available.
is-file-backed	If true Data Crow will assume that an item represents an underlying file (as is true for most of the standard modules).
is-container-managed	Indicates whether or not the item can be part of a container. *True* by default.
importer-class	The class name (+ package) which handles the file imports.
synchronizer-class	The class name (+ package) which provides the mass-update functionality.
has-depending-modules	This is for internal use only; indicates whether or not other modules are depending on this module's existence. Mainly used for the data load sequence.
default-sort-field-index	The default field used to sort the items on. This can be overruled by the user settings.

name-field-index	Indicates the field holding the name of an item (such as the title). This can be overruled by the user settings.
is-serving-multiple-modules	A powerful setting which can be used to share a module. This is for example the case for music genres. Both the audio CD and the music album module use the same instance of the music genre module. By default this property is set to *false*.

3.6 Fields

The bold properties are the ones that are the most interesting, or basic, properties.

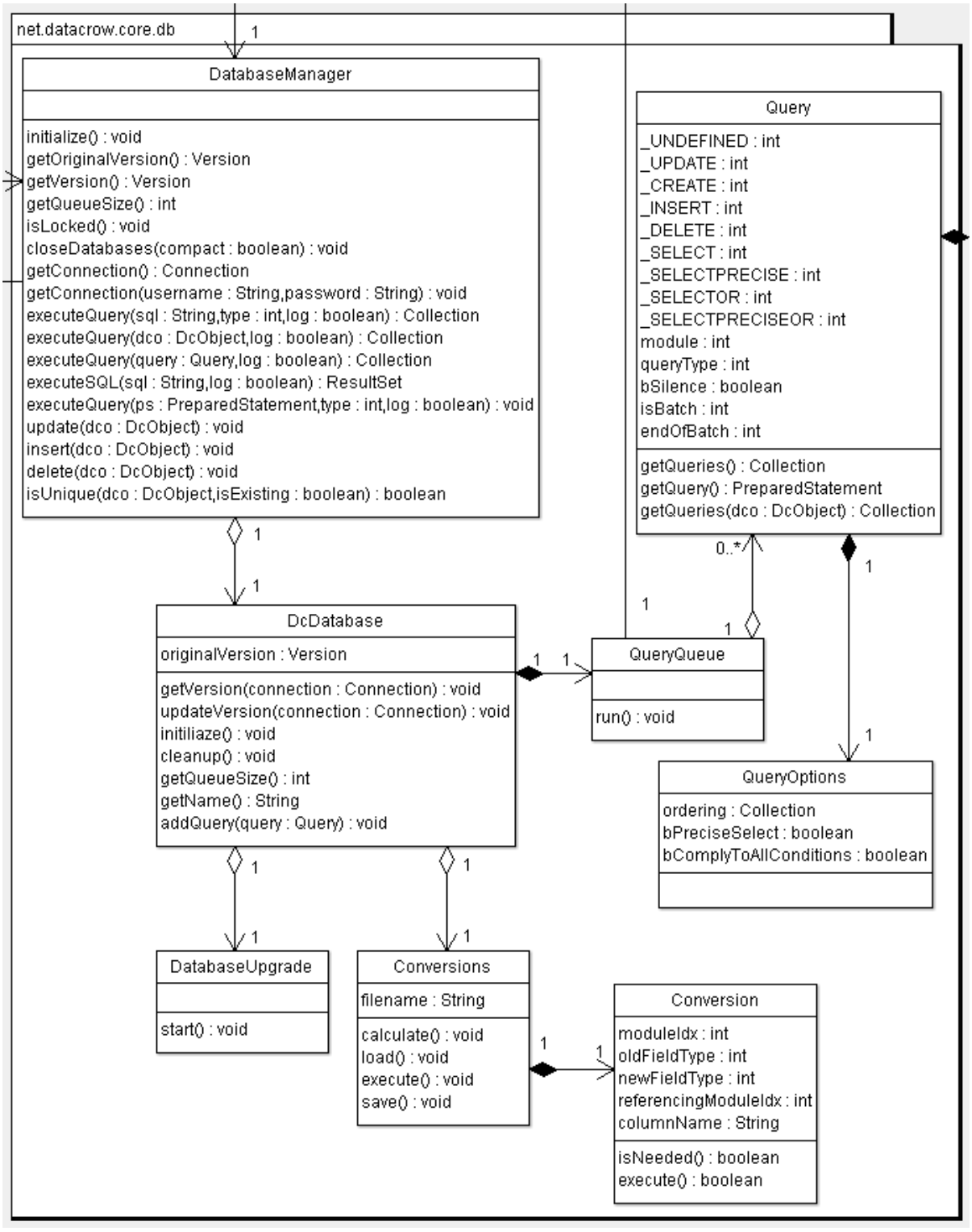
Property	Description
Index	The unique module index.
Name	The display name of this field.
database-column-name	The database column name.
ui-only	Indicates whether this field is a calculated field. When set to true the value of this field will not be stored into the database.
enabled	Is this field enabled? This can be overwritten by the user settings.
readonly	Can this field be updated by the user?
searchable	Indicates whether the user can search on this field. Typically this should be set to *false* for fields holding pictures.
techinfo	***Deprecated.*** Indicates if this field should be displayed on the technical information tab. This can be overwritten by the user settings.
maximum-length	The maximum value length.
field-type	The field type (see net.datacrow.console.ComponenFactory).
module-reference	The module index. Can either hold {index} (which means the index of this module) or should hold the module index of a referenced module. The latter is true for reference fields.
value-type	The value type. See net.datacrow.core.ValueTypes.
overwritable	Indicates if users are allowed to convert this field into a new field type by using the module alteration wizard.

4 Database Engine

Under the hood Data Crow has the HSQL engine installed. Note that normally you would not need to access the database directly as for most tasks you can use the *DataManager* class. However there are situations where direct access to the database is needed.

The *DatabaseManager* class allows you to execute SQL scripts directly on the database. You have the option to write the SQL statement yourself or to use the Query class.

4.1 UML Model


net.datacrow.core.db
DatabaseManager
initialize() : void
getOriginalVersion() : Version
getVersion() : Version
getQueueSize() : int
isLocked() : void
closeDatabases(compact : boolean) : void
getConnection() : Connection
getConnection(username : String,password : String) : void
executeQuery(sql : String,type : int,log : boolean) : Collection
executeQuery(dco : DcObject,log : boolean) : Collection
executeQuery(query : Query,log : boolean) : Collection
executeSQL(sql : String,log : boolean) : ResultSet
executeQuery(ps : PreparedStatement,type : int,log : boolean) : void
update(dco : DcObject) : void
insert(dco : DcObject) : void
delete(dco : DcObject) : void
isUnique(dco : DcObject,isExisting : boolean) : boolean
Query
_UNDEFINED : int
_UPDATE : int
_CREATE : int
_INSERT : int
_DELETE : int
_SELECT : int
_SELECTPRECISE : int
_SELECTOR : int
_SELECTPRECISEOR : int
module : int
queryType : int
bSilence : boolean
isBatch : int
endOfBatch : int
getQueries() : Collection
getQuery() : PreparedStatement
getQueries(dco : DcObject) : Collection
DcDatabase
originalVersion : Version
getVersion(connection : Connection) : void
updateVersion(connection : Connection) : void
initiliaze() : void
cleanup() : void
getQueueSize() : int
getName() : String
addQuery(query : Query) : void
QueryQueue
run() : void
QueryOptions
ordering : Collection
bPreciseSelect : boolean
bComplyToAllConditions : boolean
DatabaseUpgrade
start() : void
Conversions
filename : String
calculate() : void
load() : void
execute() : void
save() : void
Conversion
moduleIdx : int
oldFieldType : int
newFieldType : int
referencingModuleIdx : int
columnName : String
isNeeded() : boolean
execute() : boolean
1
0..*

4.2 Using the Query class

The query class is used internally by Data Crow to update, delete and insert items and also to create tables.

In the example below the Query class is used to insert a new software item with title “TEST ITEM”.

```
DcObject dco = DcModules.get(DcModules._SOFTWARE).getDcObject();
dco.setValue(Software._A_TITLE, "TEST ITEM");

try {
    Query query = new Query(Query._INSERT, dco, null, null);
    DatabaseManager.executeQuery(query, false);
} catch (SQLException se) {
    se.printStackTrace();
}
```

Note that the executeQuery method is used to execute the query. I have chosen not to log the query to the log. The Query itself can also be created using QueryOptions and Requests.

4.2.1 Query options

The QueryOptions class can be used to specify the ordering of the results. Furthermore you can specify to use exact matching (bPreciseSelect) and/or that all conditions must be met.

4.2.2 Requests

The Requests class holds instances of the IRequest interface. Requests can be added to a query. A request is a task which can be executed after the query has finished.

Requests are used as the queries are put (in general) in a queue and executed by a thread as soon as the thread has time to execute the query. To avoid that user has to wait till the query has finally been executed a (group of) request(s) can be added to the query. As soon as the query is finished the requests are processed. Requests are executed in a thread safe manner.

4.3 Using SQL statements

The following example shows how to execute a SQL statement:

```
try {
    ResultSet rs = DatabaseManager.executeSQL("SELECT * FROM SOFTWARE", false);
} catch (Exception e) {
    e.printStackTrace();
}
```

5 Plugins

Implementing a plugin requires you to at least create one class which extends the Data Crow plugin class. The class you are going to create needs to have a base package name called *plugins*. Why? Your plugin class needs to be located in the *plugins* directory, or one of its sub-directories. It so happens this is also its base package / directory.

Creating a *plugin* is simple depending on what you want the plugin to do. If you are thinking about altering item information or if you need to work with modules I advise you to read through the rest of the Data Crow developer guide.

5.1 Deploying a Plugin

Plugins and there additional libraries are contained within the *plugins* directory of the Data Crow installation.

5.1.1 The plugin class

Your plugin class needs to be copied to the plugins folder of the Data Crow installation. It is allowed anywhere within this folder. So you are allowed to create your own package structure within this base package. Just remember: base package must always be **plugins.**

5.1.2 Additional libraries

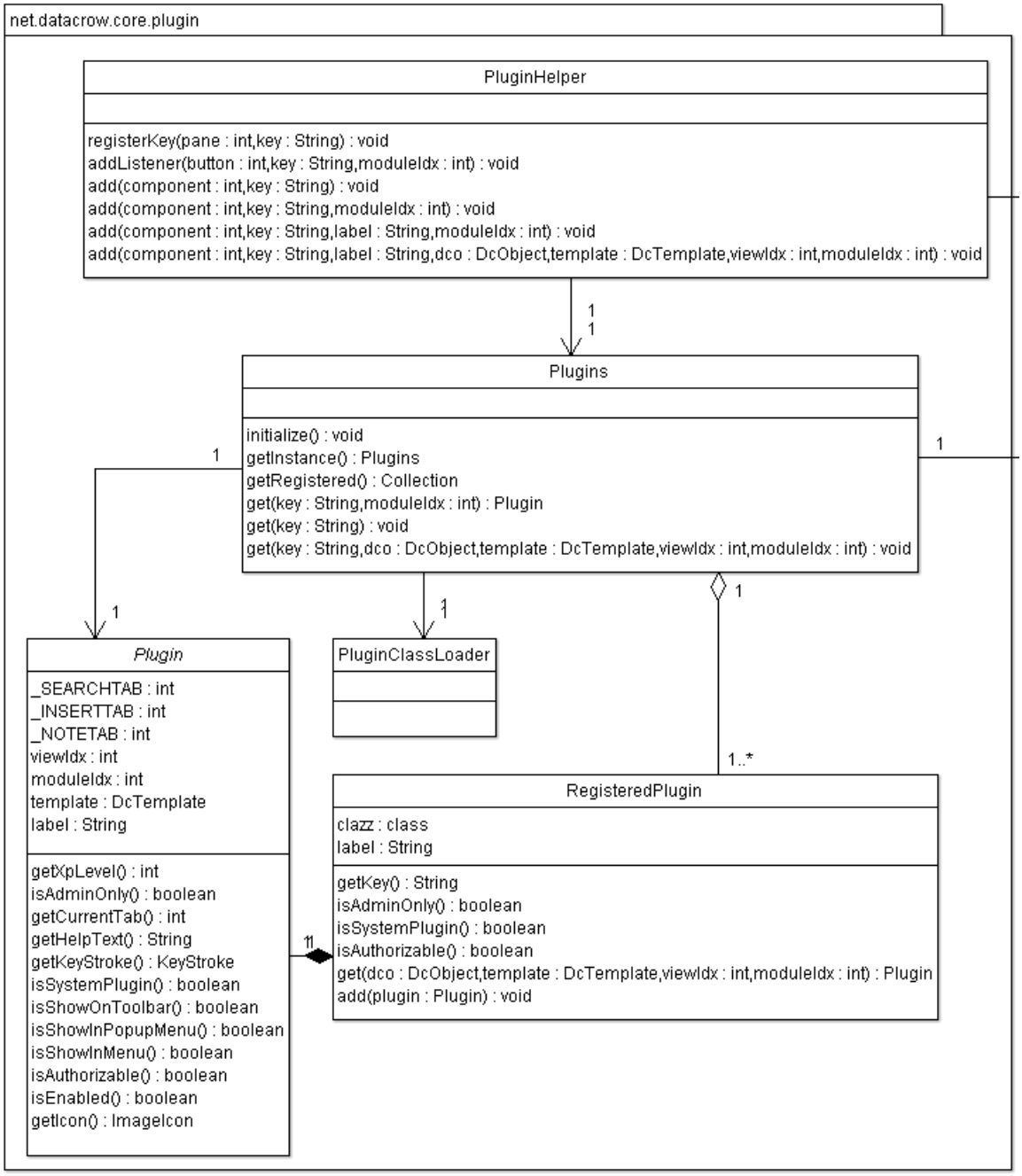
In case your software needs additional libraries to be loaded (JAR files) you can copy these to the plugins folder as well (or any of its sub folders). Data Crow will load these libraries automatically on startup.

5.2 The Plugin class

Below you'll find an overview of the methods of the plugin class (net.datacrow.core.plugin.Plugin).

Method	Description
actionPerformed	The actionPerfomed method is not something I made up but derives from the fact that the *Plugin* class implements the *AbstractAction* class from Java. This is the actual point of execution.
getIcon	The getIcon class needs you to return either *null* or an ImageIcon which will be used to represent the item in the plugin menu.
clear	Cleanup method. This method is called after the execution of your plugin has finished. You will need to override this method to remove references.
getCurrentTab	This method tells you which tab has been selected. The *MainFrame* class holds the static variables *_SEARCHTAB* and *_INSERTTAB*.
getHelpText	The value returned is used in tooltips.
getItem	Returns the currently selected Data Crow item. You will not need to overwrite this method. You will probably be using this method somewhere in your application.
getKeyStroke	Defines the key combination for starting your plugin.
getLabel	The label used to represent your plugin
getLabelShort	The short version of the label.
getModule	This method returns the currently active module, I will explain later on what you can do with this module class.
getModuleIdx	Return the current module index.
getTemplate	Some modules allow templates to be used. This method will return the currently default template or null of there is none.
getView	Delivers you the currently shown view.
getViewIdx	Returns the index of the currently used view.
isAdminOnly	Indicates if this plugin should only be available to administrator users.
isAuthorizable	Indicates whether this plugin can only be used by using having explicit rights to use this plugin.
isEnabled	Is the plugin enabled? This method should not be overridden.
isShownInMenu	Indicates whether the plugin should be displayed in the plugins menu. True by default.
isShownInPopupMenu	Indicates whether the plugin is shown in the right click menu of the view. True by default.
isShownOnToolbar	The plugin will be added to the toolbar. True by default.
isSystemPlugin	Should be set to false for custom modules. If set to true the plugin will not be added to the various menus automatically. False by default.

5.3 UML Model


net.datacrow.core.plugin
PluginHelper
registerKey(pane : int,key : String) : void
addListener(button : int,key : String,moduleIdx : int) : void
add(component : int,key : String) : void
add(component : int,key : String,moduleIdx : int) : void
add(component : int,key : String,label : String,moduleIdx : int) : void
add(component : int,key : String,label : String,dco : DcObject,template : DcTemplate,viewIdx : int,moduleIdx : int) : void
1
1
Plugins
initialize() : void
getInstance() : Plugins
getRegistered() : Collection
get(key : String,moduleIdx : int) : Plugin
get(key : String) : void
get(key : String,dco : DcObject,template : DcTemplate,viewIdx : int,moduleIdx : int) : void
1
1
1
1
1
1
Plugin
_SEARCHTAB : int
_INSERTTAB : int
_NOTETAB : int
viewIdx : int
moduleIdx : int
template : DcTemplate
label : String
getXpLevel() : int
isAdminOnly() : boolean
getCurrentTab() : int
getHelpText() : String
getKeyStroke() : KeyStroke
isSystemPlugin() : boolean
isShowOnToolbar() : boolean
isShowInPopupMenu() : boolean
isShowInMenu() : boolean
isAuthorizable() : boolean
isEnabled() : boolean
getIcon() : ImageIcon
PluginClassLoader
1
1..*
RegisteredPlugin
clazz : class
label : String
getKey() : String
isAdminOnly() : boolean
isSystemPlugin() : boolean
isAuthorizable() : boolean
get(dco : DcObject,template : DcTemplate,viewIdx : int,moduleIdx : int) : Plugin
add(plugin : Plugin) : void

5.4 Example Plugin: Hello World

```
package plugins;

import java.awt.Dimension;
import java.awt.event.ActionEvent;
import javax.swing.ImageIcon;
import net.datacrow.console.ComponentFactory;
import net.datacrow.console.components.DcDialog;
import net.datacrow.core.objects.DcObject;
import net.datacrow.core.objects.DcTemplate;
import net.datacrow.core.plugin.Plugin;

public class HelloWorld extends Plugin {
   public HelloWorld(DcObject dco, DcTemplate template, int viewIdx, int moduleIdx) {
      super(dco, template, viewIdx, moduleIdx);
   }

   @Override
   public String getLabel() {
      return "Hello World!";
   }

   @Override
   public ImageIcon getIcon() {
      return null;
   }

   @Override
   public void actionPerformed(ActionEvent ae) {
      DcDialog dialog = new DcDialog();

      dialog.setTitle("Hello world plugin");
      dialog.getContentPane().add(ComponentFactory.getLabel("Hello world!"));
      dialog.pack();

      dialog.setSize(new Dimension(400, 400));

      dialog.setCenteredLocation();
      dialog.setModal(true);
      dialog.setVisible(true);
   }
}
```

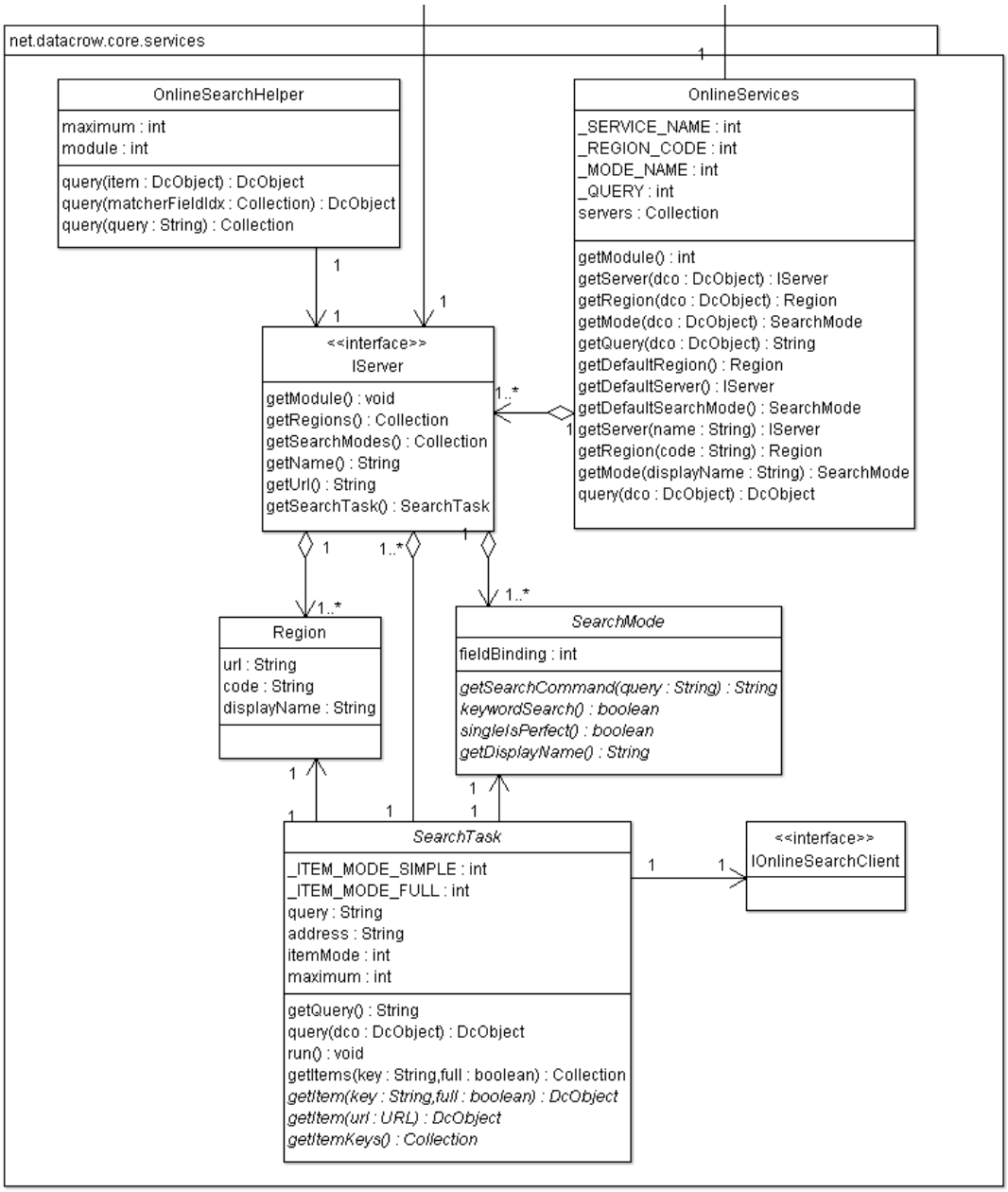
5.4.1 Result

As you can see this plugin is rather simple. Everything is done in the actionPerformed method. The compiled version of this class (HelloWorld.class) should be copied to the plugins folder of Data Crow where it will show up in the plugins menu.

6 Online Services

Online services are also pluggable. Any online service located in the *services* directory of Data Crow is loaded on startup. This makes it rather easy to write your own online service.

6.1 UML Model



6.2 The Server Definition

An online service is held together by the IServer class. Actually the IServer represents an online server such as Amazon.com or Imdb.com. The IServer class describes the available search modes (title search, ISBN search, search 1, search 2) and, if applicable, the available regions (such as Amazon.co.uk, Amazon.de). Both the search modes and the regions are optional.
The IServer class is also responsible for providing the actual search task.

6.2.1 Search Modes

A search mode must extend the abstract *SearchMode* class. The most important method of this class is the *getSearchCommand* method. The method takes the query as a parameter and is expected to return an actual URL or an addition which can be added to the URL of the server or selected region. This depends on your own implementation!

```
public class KeywordSearchMode extends SearchMode {

   private final String searchIndex;
   private final String label;

   public KeywordSearchMode(String searchIndex, String label, int fieldBinding) {
      super(fieldBinding);
      this.searchIndex = searchIndex;
      this.label = label;
   }

   public String getDisplayName() {
      return label;
   }

   public String getSearchCommand(String s) {
      String cmd =
          "Operation=ItemSearch" + "&SearchIndex=" + searchIndex + "&Keywords=" + s;
      return cmd;
   }

   public String getSearchIndex() {
      return searchIndex;
   }

   @Override
   public boolean keywordSearch() {
      return true;
   }
}
```

6.2.2 Regions

When the online service can be found in several regions (such as .com, .de and .nl) you might want to consider making all these instances or regions available to the user. You do not have to implement or create your own region class. Instead you will register these in your server class.

An example, as taken from the Amazon.com implementation:

```
private Collection<Region> regions = new ArrayList<Region>();

public AmazonServer() {
   regions.add(new Region("us", "United States (english)", "http://webservices.amazon.com"));
   regions.add(new Region("de", "Germany (german)", "http://webservices.amazon.de"));
   regions.add(new Region("fr", "France (french)", "http://webservices.amazon.fr"));
   regions.add(new Region("ca", "Canada (english)", "http://webservices.amazon.ca"));
}

public Collection<Region> getRegions() {
   return regions;
}
```

As you can see the region is solely used to replace the base URL.

6.2.3 Search Task

The search task performs the actual parsing of the online server information. The implementation is based on the assumption that there is an overview page from with item keys can be parsed. In the second step these keys will be used to go to the detailed information page for each of the items.

To create the Search Task you have to implement the SearchTask class. The following methods need to be implemented:

Method	Description
getItemKeys(URL)	Should parse the keys or item details URLs from the online service. You are totally free on the implementation of this. Make sure to form the correct URL where you are going to retrieve the information from (by using the *SearchCommand*, *Server* and *Region* classes).
getItem(URL)	The URL to the detailed item information should be passed to this method. This method is used by the Synchronizer class (aka. The mass update feature). Just make sure to implement this correctly. It is advisable to let the other getItem methods call this method. Both for reusability and good coding practices.
getItem(String key, boolean full)	The keys as created / parsed by the *getItemKeys* method are passed one by one to this method. Form a fully functioning URL pointing to the detailed information page of the item. Use this URL to call the *getItem(URL)* method.

6.2.4 Putting it all together in the server definition

The only thing remaining, before the service can be deployed, is creating a class implementing the *IServer* interface.

Method	Description
getModule	Indicates for which module you have designed your service. Return the index of your module here (for example DcModules._MOVIE).
getName	The display name of the server.
getRegions	Collection of the available regions.
getSearchModes	Collection of the available search modes.
getUrl	The base URL of the server. Note that, depending on your implementation, the URL of the region should be used.
getSearchTask	Creates a new instance of the SearchTask class (your own SearchTask implementation).

6.3 Deploying an Online Service

Package your class files as a JAR file and copy it to the *services* directory. Any additional libraries you might be using should also be copied to this directory. All JAR files from this directory are loaded automatically on startup.