



i-net
Crystal-Clear**X** version 10

Programming Guide

Server

Table of Contents

1 Overview.....	3
1.1 Call Sequence.....	3
2 ReportServlet.....	5
3 Cache.....	6
4 EngineFactory.....	7
5 PropertiesChecker.....	8
6 Database.....	9
6.1 Setting the Database class.....	9
6.1.1 Data Source Configuration.....	9
6.1.2 API.....	10
6.2 Implementation of the Database class.....	10
7 Dummy JDBC Driver.....	11
7.1 Building your own JDBC driver.....	11
8 EngineRenderData.....	12
9 CacheRenderData.....	13
10 RDC and Engine.....	14
11 Compare the programming techniques.....	15
12 Final Recommendation.....	16

1 Overview

I-net Crystal-Clear contains an extensive API for programming the report server. This guide will give you an overview about the possibilities. More complete information can be found in the [documentation](#), e.g. the [list of libraries](#) and the [API documentation](#). You can find the documentation in the installation directory (if installed) or on the [i-net Crystal-Clear website](#).

1.1 Call Sequence

To help you understand where you can change what, we'll show you first the sequence of code points of a typical report request. Many of these methods (as for example the PropertyChecker methods) can be overloaded by your own classes.

- ▷ ReportServlet.service()
This is the entry point for the Servlet Engine
- ▷ ReportServlet.afterPropertiesStoredHook()
This is called after the POST and GET parameters was read and encoded.
- ▷ Static content handling like the Applet jar file
- ▷ ReportServlet.checkHtmlPageProperties()
PropertyChecker.checkHtmlPageProperties()

Executed only if a HTML page for the Java viewer applet is to be generated (e.g. if a report is being requested as a Java viewer report for the first time by a client).
- ▷ Cache.getKey()
Request a report key for the parameters
- ▷ PropertiesChecker.checkProperties(Properties,...)
Report name has been turned into an absolute path, some properties were converted. This is the last possibility to set or modify the property values before the report is run.

- ▼ EngineFactory.getKey()
A report key is generated from the relevant properties.
- ▼ EngineFactory.createEngine()
If the data for the report is not in the cache then a new Engine will be created at this point.
- ▼ PropertiesChecker.checkProperties(Engine,Properties,...)
Now the method with parameter “Engine” is called. At this point you can modify the Engine before it is run but you can't change the property values because the report key has already been generated.
- ▼ Engine.execute()
The Engine will render the report and the data of the rendered report are saved asynchronously in the Cache.
- ▼ Database.getReportData(Engine,String)
The data for the report is requested.
 - ▼ Engine.setData()
The Database class sets the engine's data.
- ▼ Cache.getPageAndWait()
This method waits until the requested page becomes available, while asynchronously, the report is rendered.

2 ReportServlet

If you deploy i-net Crystal-Clear in a servlet engine then the class `com.inet.report.ReportServlet` is the main entry point. If you only want to control the access to the reports then the easiest solution is to extend this `ReportServlet`. You can find the [Report Servlet API](#) in the [API documentation](#).

There are some methods which are called on the live cycle of the report request.

You can:

- ▼ set or modify the report name
- ▼ set or modify the report parameters (prompts)
- ▼ modify the HTML page of the applet
- ▼ modify the Engine after the template was read
- ▼ use a specific [output format](#) (applet, PDF, etc.)
- ▼ use the Remote Configuration Manager

You can not:

- ▼ create the Engine yourself

3 Cache

For some output formats like the Applet Viewer or HTML the server will send the rendering result in multiple steps. Therefore it is necessary to cache the results between different request. Also, with the cache it is possible to share the rendered results between multiple users. This optional feature can reduce the load on the server and it can be enabled/disabled using the Configuration property: “[Reload On New Request](#)”.

You can find the [cache API](#) in the [API documentation](#).

The following sample shows a simple usage of the Cache API to export to PDF:

```
Cache cache = Cache.getCache();
Properties reportProperties = new Properties();
reportProperties.put("report", "report1.pdf");
ReportCacheKey key = cache.getKey(reportProperties);

int pageCount = cache.getPageCountAndWait(key);
for (int i = 1; i <= pageCount; i++) {
    byte[] = cache.getPageAndWait(key, i);
    ....
}
```

More samples for the usage of the Cache API you can find in the Java code samples directory<: “<installation directory>/Documentation/sample/cache”.

If you use the Cache API directly you do not need to implement the caching itself. You have the full control over the connection between client and server.

You can:

- ▾ set or modify the report name
- ▾ set or modify the report parameters (prompts)
- ▾ modify the HTML page of the applet
- ▾ create the Engine (limited with addEngine)

You can not:

- ▾ use the Remote Configuration Manager

4 EngineFactory

The Cache API has 2 static methods `getEngineFactory` and `setEngineFactory`. With these methods you can register your own global EngineFactory.

You can find the [EngineFactory API](#) in the [API documentation](#).

With the EngineFactory you can:

- ▷ use your own report templates
- ▷ use your own parameters
- ▷ use the Remote Configuration Manager
- ▷ control the Engine creation
- ▷ skip some parameters from the report key

You can not:

- ▷ modify the HTML page of the applet

5 PropertiesChecker

Another option is to implement the PropertiesChecker interface.

You can find the [PropertiesChecker API](#) in the [API documentation](#).

There are multiple solutions to set a custom PropertyChecker:

- ▷ set the class name in the [property “Property Checker”](#) of the currently used Configuration (recommended)
- ▷ use a class name “PropertiesChecker” (without package)
- ▷ call ReportSocket.setPropertyChecker before ReportServlet.init() was called

With the PropertyChecker you can accomplish much the same as with an overridden ReportServlet, though ReportServlet has more interaction points.

You can:

- ▷ set or modify the report name
- ▷ set or modify the report parameters (prompts)
- ▷ modify the HTML page of the applet
- ▷ modify the Engine after loading
- ▷ force a specific output format (applet, pdf, etc.)
- ▷ use the Remote Configuration Manager

You can not:

- ▷ create the Engine

6 Database

If you want to manually set the data for a report then you can do it with any of the `Engine.setData()` methods. An alternative method is to set a data factory.

An instance of the class `com.inet.report.Database` is the factory for the report data. Most implementations of the Database use JDBC but it is also possible to set the data without JDBC. The advantages of the Database class is that it is easier to set the data for sub reports. This is the only solution for setting individual data for each sub report.

6.1 Setting the Database class

There are multiple possible solutions.

6.1.1 Data Source Configuration

The easiest solution is to set the class name of the Database class in the [Data Source Configuration](#) on which the report is based.

Note: A Data Source Configuration with the same name can have different underlying data sources on the server machine and the machine running i-net DesignerXML.

Possible problems with this approach could be that:

- ▷ an administrator changes, renames or deletes the data source configuration pointing to your Database implementation
- ▷ the Database class can not be loaded

In these cases i-net Crystal-Clear does its best to locate the data source via fallback solutions.

6.1.2 API

Another solution is to set your Database class via API. You need access to the Engine instance after loading. To get the Engine instance you need to use one of the other described programming techniques.

```
Engine engine = ...
DatabaseTables dbt = engine.getDatabaseTables();
for (int i=0; i<dbt.getDatasourceCount(); i++) {
    Datasource ds = dbt.getDatasource(i);
    ds.setDatabase(myDatabase);
}
```

The same must be repeated for each sub-report.

6.2 Implementation of the Database class

If you want to implement your own Database class then you need to extend from `com.inet.report.Database`. Specifically, you need to override the following methods:

▷ `getReportData(Engine,String)`

This method is called for Engine and sub-report Engines separately. You need to call one of the `Engine.setData()` methods.

▷ `useJdbcDriver()`

Override this method if you do not use JDBC to get the data.

▷ `getReportDataPerInstance()`

This method is important for the data of sub-reports.

7 Dummy JDBC Driver

If you want set the data via API you can also use your own JDBC driver. JDBC is the standard data factory of Java. At first glance, this could seem to be very difficult because a JDBC driver has a large API. But i-net Crystal-Clear only uses a few of the API methods. You can find a sample for a dummy JDBC driver in the directory: “<installation directory>/Documentation/sample/jdbc_YourOwnDriver”.

You can set the used JDBC driver in the Data Source Configuration used by your reports. The JDBC driver can be used at design time (in the report designer) and at runtime in the server. A dummy JDBC driver is the only solution to setting the data via API, designing a report with your own data, and using a [Row Buffer](#).

7.1 Building your own JDBC driver

Look at the samples in the folder “<installation directory>/Documentation/sample/jdbc_YourOwnDriver”. There are 9 Java source files. Copy these files in your Java IDE and change the implementation of the implemented methods. Create a jar file and add it to the lib directory of the i-net DesignerXML.

Now you can design a report using your own JDBC driver.

8 EngineRenderData

If you want to implement an application without web interface or your own client-server communication protocol for example CORBA or SOAP then a possible solution is to extend the class `RenderData` on the Java Viewer (client) side and the class `EngineRenderData` on the server side. The most important method is `createEngine(Properties)`. This is the factory method for the Engine. You need to transfer all method calls from the `RenderData` on client and back. This class does not use the cache. The Engine and the rendered data is hold in the memory. The i-net DesignerXML uses this class for generating data for its embedded report viewer.

You can:

- ▶ use all features of the Java viewer
- ▶ have full control over the Engine creation
- ▶ have full control over the client server communication

You can not:

- ▶ use the Remote Configuration Manager
- ▶ create reports larger than the RAM allows, since the report results are held in memory.

9 CacheRenderData

The CacheRenderData is an alternative implementation of the RenderData interface. The difference to the EngineRenderData is that it uses the cache. Therefore you can't create the Engine itself.

You can:

- ▷ use all features of the Java viewer
- ▷ have full control over the Engine creation
- ▷ have full control over the client server communication
- ▷ create reports larger than memory-only

You can not:

- ▷ use the Remote Configuration Manager

10 RDC and Engine

The classes `com.inet.report.RDC` and `com.inet.report.Engine` are the starting points if you want to work with the low-level API directly. If you use the low-level API then you will have to do many things like caching and client server communication yourself.

You can:

- ▶ fully control the Engine creation
- ▶ fully control the client server communication

You can not:

- ▶ use the Remote Configuration Manager
- ▶ create reports larger than your RAM can hold
- ▶ use all features of the viewer (search, error messages).

11 Compare the programming techniques

	ReportServlet	Cache	EngineFactory	PropertyChecker	Database	Own JDBC Driver	EngineRenderData	CacheRenderData	RDC and Engine
All Viewer Features 1)	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	No
Report Cache	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes	No
Full control over the Engine creation	No	limited 2)	Yes	No	No	No	Yes	No	Yes
Modify report parameters	Yes	Yes	No	Yes	No	No	Yes	Yes	Yes
Modify HTML page	Yes	Yes	No	Yes	No	No	Yes	Yes	Yes
Full control over the client server communication	limited 3)	Yes	No	No	No	No	Yes	Yes	Yes
Complexity (1 - simple, 5 - difficult)	1	3	2	1	1	3	4	4	5
Modify the Engine after loading	Yes	limited 2)	Yes	Yes	Limited 4)	No	Yes	No	Yes
Set the report data	limited 5)	No	limited 5)	limited 5)	Yes	Yes	limited 5)	No	limited 5)

1) The error handling and the search feature is not always possible

2) With `Cache.addEngine(Engine)` no large reports are possible

3) You can call the `ReportServlet` with your own implementation of `HttpServletRequest` and `HttpServletResponse`.

4) In `getReportData` you get an `Engine` and you can do some modifications on the `Engine` but it is not recommended because the rendering has already started.

5) You can not set the data for each individual sub report.

12 Final Recommendation

Our recommendation is to use ReportServlet, EngineFactory or both together. If you only want to set the data then your own Database class is the best solution.