Java Engine User's Guide

KDCalc™ (Patents Pending)

Excel-Compatible Spreadsheet Engine for Java
Version 4.2.x

Knowledge Dynamics, Inc.

"The Killer-App for Software Development"™

Note: See *KDCalc Designer User's Guide* for Excel Plug-in Help.

Welcome to KDCalc™ V4.2.x by Knowledge Dynamics®, Inc. (Patents Pending)

"The Killer-App for Web Development"™

What is It?

KDCalc is a Microsoft Excel® compatible spreadsheet designer plugin and a runtime engine for Java and .Net.

Why do I need it?

With KDCalc, non-programmers can design, build, and test complex logic in Excel, then embed it into Web applications of all kinds. KDCalc can be paired with 3rd party grid and graph controls for lightweight, interactive web presentations. KDCalc also supports a technique called 'Spreadsheet Programming', where spreadsheets are used in transaction processing or other server side-processing. KDCalc Modules can be deployed to any platform with a JVM or the .Net CLR. Excel is not needed on the end-user's machine.

What do I get?

KDCalc consists of an Excel plug-in that parses Excel workbooks and emits a compressed file format, a runtime engine that support most built-in Excel functions, and an API that allows consistent access to your spreadsheets' functionality.

How do I use it?

The KDCalc plug-in installs right into the Microsoft Excel® toolbar. Click a few buttons to set up the compilation options, and KDCalc creates a compressed file that performs the calculations and data transformations in your workbook.

Access the spreadsheet data and calculations through simple API calls. Put values into data cells and get results from formula cells. Navigate Sheets by Name or Index. Access Cells and Ranges by Name or by Row, Col.

System Requirements

The KDCalc V4.2.x Designer Plug-In requires the following:

- Microsoft Windows 95, 98, NT, 2000, or XP
- Microsoft Excel 2000 or XP for Windows
- 64MB RAM
- 15MB Free Hard Drive Space

The KDCalc V4.2.x Engine runtime requires a JVM compatible with JDK 1.1 or higher, or the .Net CLR.

Licensing

The following is a brief overview of the KDCalc V4.2.x licensing terms. See the attached End-User License Agreement for the full license.

For development KDCalc V4.2.x is licensed *per developer*. Each Developer that writes code that will directly or indirectly call the KDCalc runtime engine must obtain a license. A Licensee is not permitted to generate modules for other developers unless the other developers also have properly obtained licenses.

For individual deployment KDCalc V4.2.x Engine is royalty-free for deployment in applications that execute in personal computer environments, such as stand-alone applications or applets in a web browser.

For server deployment KDCalc V4.2.x Engine Server Edition is licensed *per CPU* for deployment in applications that execute in server environments such as Web Services, EJBs and Web Application Servers.

For embedded systems deployment KDCalc V4.2.x Engine Embedded Edition is licensed *per application* for deployment in applications that execute in embedded systems environments such as PDAs, embedded microprocessors, cell phones, etc.

Table of Contents

Installation and Registration

Development System Requirements

The KDCalc V4.2.x Designer Plug-In requires the following:

- Microsoft Windows 95, 98, NT, 2000, or XP
- Microsoft Excel 2000 or XP for Windows
- 64MB RAM
- 15MB Free Hard Drive Space

Java Execution System Requirements

The KDCalc V4.2.x runtime engine requires the following for Java deployment:

- Java JRE 1.1 or higher

Installation

To install KDCalc V4.2.x, make sure Excel is not running and simply double-click on 'setup.exe'. It installs like any other Windows application.

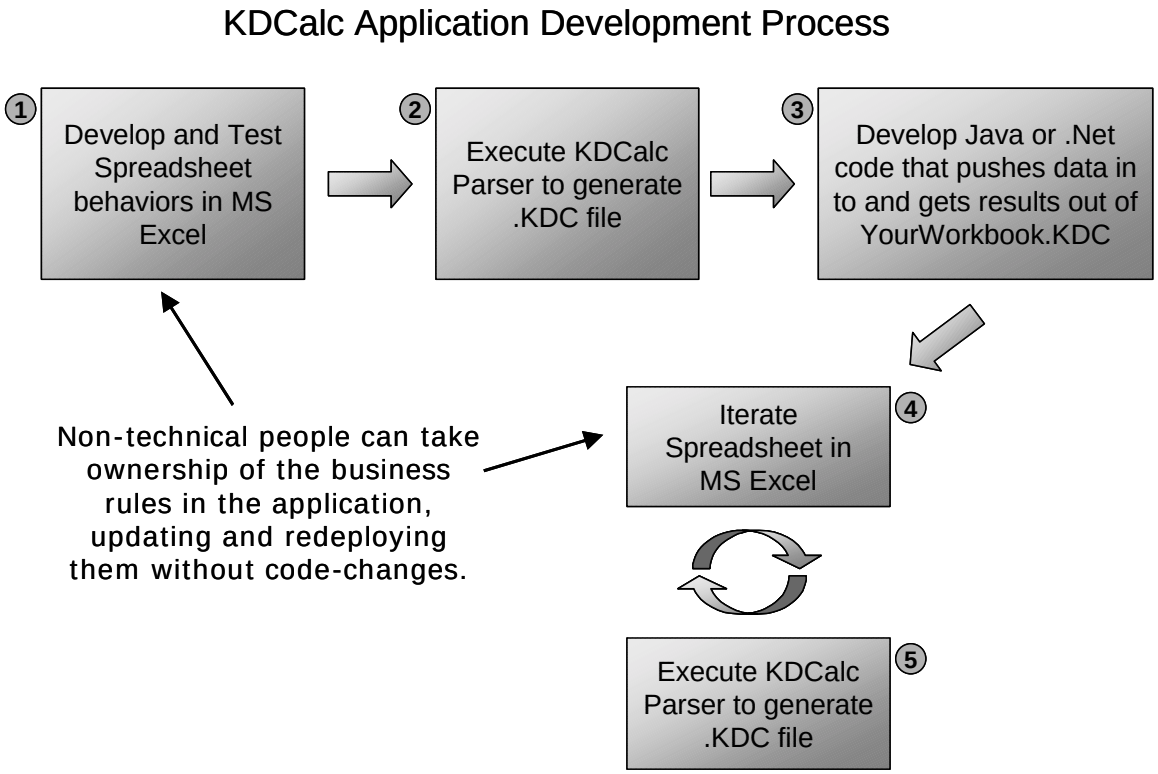
Registration

The KDCalc Excel Plug-In installs as a size-limited demo. It will only compile cells A1-E10 on the first 2 Worksheets. KDCalc must be registered to unlock its full functionality.

To register KDCalc, click the 'About / Register' button on the main KDCalc window. Enter your License Code and registration information. An Internet connection is required.

Developing with KDCalc (The Process)

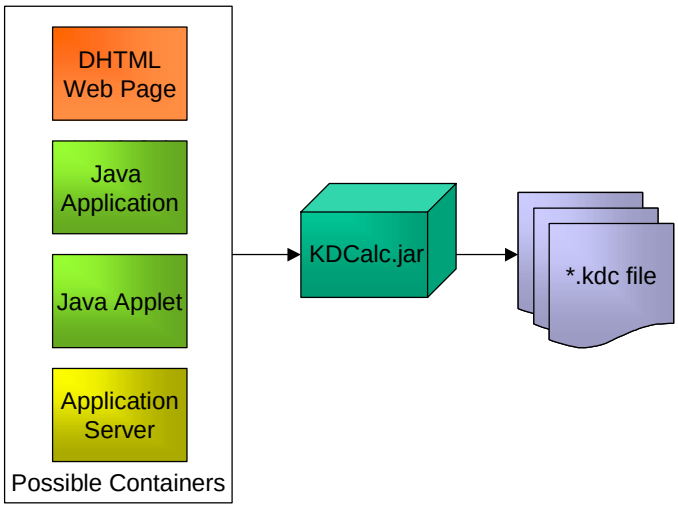
Developing Applications with KDCalc is fast, fun, and easy!



1. KDCalc lets you use the familiar, productive environment of Microsoft Excel to develop complex calculations and data transformations.
2. With a few mouse clicks, KDCalc turns your Excel cell formulas into a compressed, compiled executable format. The business rules in the spreadsheet are duplicated in the .KDC file with no chance of miscommunication between designers and developers.
3. All you have to write is the plumbing code that moves data in and out of KDCalc cells. This might be a GUI that gets data from the user, pushes it into the KDCalc Engine, pulls the results from the KDCalc Engine, and displays the results in a graph or table. It could also be an OLAP application running on a server, extracting data from a database and publishing the results to a web report. The 'setCellValue' and 'getCellValue' API methods make this integration simple. The list of potential applications is endless.
4. Once the application is built, business people can modify and enhance the spreadsheet in Excel and as often as desired.
5. The new business rules can be deployed to production without any code changes or recompilation of the application. The engine is small (about 75K) and support over 180 Excel functions. .KDC files are generally about ¼ the size of the original .XLS file.

Using KDCalc Engine for Java

The KDCalc Calculation Engine is in KDCalc.jar. This jar contains the class CalcEngine which is the primary public interface to interact with KDCalc. The KDCalc Engine can be used in a variety of configurations as illustrated below. In addition, future versions of KDCalc will be J2ME compliant and run on PDA's and cell phones.



Sample Code:

Usage:

```
// to load the calculation engine:
try{
   CalcEngine cEng = CalcEngine.getCalcEngine("tankGame.kdc");
}catch(IOException ioe){}
// simple get and set commands:
String s = cEng.getText(2,2);
double d = cEng.getNumber(3,2);
cEng.setValue(3,3, "hello");
cEng.setValue(32,2, true);
// getting a range:
Range r = cEng.getRangeFromName("NamedCell");
…
```

Saving and Specifying State:

KDCalc can easily store/persist its state to XML. The API methods *getState*, *saveState*, *loadState*, and *processXMLState* are the main methods involved with persistence.

Best Practice: Use KDCalc in an application to store the application's state. Map each control to a cell in KDCalc. For example if a checkbox is named "process", create a corresponding named cell in Excel named "process". KDCalc the Excel file.

In the application, when the checkbox is checked, call the KDCalc API to set the value of the "process" named cell to *TRUE*. When the user saves or exists the application, use the state methods to store in a database or on the file system. When the user returns to the application, just reload the KDCalc

state from the file or database. To restore the user's state, get the value out of the named cell "process" and set the state of the corresponding checkbox.

Furthermore, you can subclass existing controls or use a decorator pattern to automatically associate different controls with named cells and ranges.

In order to save KDCalc's state, call the *saveState* or *getState* methods as in the following example:

```
File f = new File("save.xml");
cEng.saveState(f);
…
// or
String s = cEng.getState();
// write the String to a database
…
```

When using the *saveState* or *getState* methods, only cells which were changed at runtime are actually persisted out. The XML document below is what would result from calling *getState* on a kdc file where only two cells were changed.

```
<workbook n="tankGame" v="3.0.0">
      <sheet n="basic" p="1">
            <c r="57" c="6" t="1" v="9.9"/>
            <c r="6" c="7" t="2" v="hi"/>
      </sheet>
</workbook>
```

The above XML states that a workbook named "tankGame" had a sheet called "basic" at position 1. On this sheet, two cell values were saved. A cell at (57, 6) saved out a number (type 1) value of 9.9, while cell (6,7) saved out a text (type 2) value of "hi".

An XML schema for describing KDCalc's XML state is located at: "*.\lib\KDCalcXMLState.xsd*". The following table summarizes the possible XML elements and attributes for saved state:

saveState / loadState XML Element and Attributes Tags

Tag	Definition
workbook	*Worbook Element*
n	Name of the workbook.
v	Version number of KDCalc that this XML was generated by/for.
sheet	*Sheet Element*
n	Name of the sheet
p	Position / index of the sheet
name	*Name Element*
n	The name of the named cell or named range.
c	*Cell Element*
r	Row number
c	Column number
t	The value type of the cell. See **Types** table.
v	Value to be passed in

Note: All attribute names are only one character long in order to allow for an extremely fast and small XML parser. KDCalc's internal XML parser is non-validating. Undetermined problems can result from

malformed XML documents. If the XML being sent to KDCalc needs validating, it is recommended to pass the XML through a validating parser with the included schema.

A list of KDCalc's available types is shown below:

Types

Constant	Number	Definition
NUM	1	Number
TXT	2	Text / String
BOOL	3	Boolean / Logical Value
ERR	4	Error
DATE	5	Date

Note: The *Constants are* available as public properties on the CalcEngine class.

In Excel some cells have formulas, others are blank, while others contain numbers, text, Boolean values, or dates. In addition, some of these types may be coerced with formatting. KDCalc stores cell values by their type as categorized in the table above. The 't' attribute is responsible for enforcing this typing. The following table shows the results of what can happen when mixing the different types:

Cell Type	*text*	*number*	*boolean*	*error*	*date/#*
Example	***hello***	***3.14159***	***TRUE***	***#N/A***	***3/2/1974***
Get as NUM	NaN	3.14159	1	NaN	27090
Get as TXT	hello	3.14159	TRUE	#N/A	27090
Get as BOOL	FALSE	TRUE	TRUE	FALSE	TRUE
Get as ERR	hello	3.14159	TRUE	#N/A	27090
Get as DATE	1/0/1900 0:0:NaN	1/3/1900 3:23:53.38	1/1/1900 0:0:0.0	1/0/1900 0:0:NaN	3/2/1974 0:0:0.0

In order to re-load the CalcEngine to from a persisted XML file, the associated kdc file would first be loaded and then *loadState* would be called. The following code snippet demonstrates this:

```
cEng.getCalcEngine("c:\\temp\\tankGame.kdc");
File f = new File("save.xml");
cEng.loadState(f);
…
```

In a different scenario, the XML could be loaded from a database and the application could call loadState with the XML string. An important point to notice is that the data can be populated into any .kdc file. This means that data can be saved out of one .kdc file and populated into another .kdc file for purposes such as testing different data sets.

The *zCPersist* option in the Designer may be used for finer control over which cells are actually persisted. All cells marked with the *zCPersist* color schema as described above in the KDCalc Designer section are saved out to XML whether they have changed or not.

The XML used to populate KDCalc may be written by another process or created by hand. This may be useful as a way of populating KDCalc from a database where calling the KDCalc API directly is not an option.

See the Tank Game Application for a working example of using KDCalc's XML persistence. The File|Load and File|Save menu commands work using these API's.

Best Practice: When using KDCalc in production applications it is wise to use the String versions of *loadState* and *getState*. Using these API's allows you to zip and encrypt the Strings in memory before saving the state to disk.

ProcessXMLState API:

A more advanced persistence option is available via the *processXMLState* API. This method provides more control over both input and output of data. With a single call to process XML state, you can pass in multiple cell values and retrieve multiple results all at once.

For this API, the XML contains two sections, an <*input*> section and a <*outputSpec*> section. The <*input*> section contains all of the values to be passed into the spreadsheet. It can contain named ranges, named cells, ranges, and cells. When using named ranges and named cells, all of the values are set to type specified by 't' and the value specified by 'v'. If a value is set on a cell which contains a formula, or other cell value, it is overwritten with the new value of the specified type. Note: The API does not throw an error when overwriting a formula cell.

The <*outputSpec*> section specifies which cells will be returned from KDCalc as an XML document. If the type ('t') is specified in the <*outputSpec*> section on a cell, then the returned type is type-cast to the type specified. If the type is omitted, the last set type of the cell is returned. Values 'v' are not associated with elements in the <*outputSpec*> section.

Either <*input*> or <*outputSpec*> sections can be omitted if you just want to set values or only want to return specified values, respectively.

An example input XML for the processXMLState is shown below with a mixture of the different elements and attributes:

Specified State XML:

```
<processXML v="3.0.3">
	<input>
		<name n="namedCell" t="2" v="hi"/>
		<name n="namedRange" t="1" v="1.1"/>
		<sheet n="Sheet1" p="1">
			<c r="5" c="2" t="1" v="42"/>
		</sheet>
		<sheet n="Sheet2" p="2">
			<c r="6" c="2" t="5" v="12:31:02"/>
		</sheet>
	</input>
	<outputSpec>
		<name n="namedCell" t="2"/>
		<name n="namedRange"/>
		<sheet n="Sheet1" p="1">
			<c r="10" c="2" t="1"/>
			<c r="1" c="2"/>
			<range r="39" c="7" s="88" d="7" t="1">
			<range r="39" c="8" s="88" d="9">
		</sheet>
		<sheet n="Sheet2" p="2">
			<c r="1" c="1" t="2"/>
		</sheet>
	</outputSpec>
</processXML>
```

To run this in KDCalc use code like the following:

```
File f = new File("specState.xml");
String retStr = cEng.processXMLState(f);
```

When the above specified XML state is run through a simple spreadsheet, it could produce the following output:

Output:

```
<workbook n=" test1" v="3.0.3">
        <name n="namedCell">
           <sheet n="Sheet1" p="1">
                <c r="2" c="2" t="2" v="hi"/>
           </sheet>
        </name>
        <name n="namedRange">
           <sheet n="Sheet1" p="1">
                <c r="3" c="2" t="1" v="1.1"/>
                <c r="4" c="2" t="1" v="1.1"/>
           </sheet>
        </name>
        <sheet n="Sheet1" p="1">
                <c r="10" c="2" t="1" v="2.9"/>
                <c r="11" c="2" t="2" v="test"/>
        </sheet>
        <sheet n="Sheet2" p="2">
                <c r="1" c="1" t="2" v="hello"/>
        </sheet>
</workbook>
```

The output is the same format as the XML output from the saveState/getState methods with the addition of one element: the *<name>* element which is used to wrap the cell/range it contains. In addition, note that all ranges and named ranges are resolved into their individual cells. Since the output is the same as that from saveState/getState, it can in turn be loaded back up as state into KDCalc using the *loadState* method.

A schema for describing the processXML document is located at: "*.\lib\ProcessXMLRequest.xsd*". The table below summarizes the different XML Elements and Attributes used in a Specified State XML document:

Specified State XML Element and Attributes Tags:

Tag	Definition
processXML	*Element denotes an XML document for the processXMLState method*
v	Version number of KDCalc that this XML was generated by/for.
input	*Input Element for Specified State API*
sheet	*Sheet Element*
n	Name of the sheet
p	Position / index of the sheet
name	*Name Element*
n	The name of the named cell or named range.
t	The type to be set or retrieved. See **Types** table.
v	A string representation of the value to be passed in.
range	*Range Element – Range elements are for input only*

r	The starting row for the range.
c	The starting column for the range.
s	The ending row for the range. (why 's'? 's' = 'r'+1)
d	The ending column for the range. (why 'd'? 'd' = 'c'+1)
t	The type to be set or retrieved. See **Types** table. All values in a range are retrieved or set to this type. If omitted, each vaue is returned with its natural type.
v	Value to be passed in – valid for input only. In a range, all values in the range are set to this value.
c	*Cell Element*
r	Row number
c	Column number
t	The type to be set or retrieved. See **Types** table.
v	Value to be passed in.
outputSpec	*Output Specification Element for Specified State API*

The *processXMLState* API can be especially useful for web services and server-side processes. Instead of calling individual *setNumber*/*setText* and *getNumber*/*getText* API calls, they can be batched up and sent as XML all at once.

KDCalcApplet:

KDCalc.jar contains a wrapper Applet called KDCalcApplet. This Applet allows KDCalc.jar to be embedded in a web page to create highly dynamic and interactive web pages – this means that ***no round trip to a server is necessary for calculations***. KDCalcApplet allows you to load up a .kdc file from within an HTML page and call it from javascript.

Since Javascript is an untyped language, the API for communicating to the CalcEngine through javascript is different. The KDCalcJSAdapter class provides the API for communicating to the CalcEngine.

Below is a code snippet that shows how to embed KDCalc within an HTML page:

```
<applet mayscript name="KDCalcApplet" archive="kdCalc.jar"
code="kdyn.calc.KDCalcApplet" width="1" height="1">
    <param name="FileName" value="savingsCalculator.kdc">
</applet>
```

Sample Javascript code:

```
<script language="Javascript”>

var calcEng1;
var calcEng2;

function init()
{
       calcEng1 = KDCalcApplet.getKDCalcJSAdapter();
       calcEng1.setNumberD(1,1,42.32);
       calcEng1.setNumberS(1,2,”123.93”);

       calcEng2 = KDCalcApplet.getNewKDCalcJSAdapter(“test.kdc”);
       calcEng2.setValueS(“Sheet2”, 1, 1,”6.9636”);
```

```
}

</script>
```

See the Investment Calculator sample for details on using KDCalc within HTML. See the javadoc api found in the *docs/api* folder for more method details.

Several methods such as *getNumberStringFrom1DRange* are available as well because Arrays cannot be passed back and forth between Java and Javascript. These methods pass back strings which are delimited with "|" and "~" characters that can be transformed into arrays within javascript. See the API documentation for details and example code.

Additionally, you can change or add additional ways of communicating to the CalcEngine by wrapping it with your own Applet wrapper.

KDSpreadSheetControl:

KDCalcTools.jar contains a javabean named KDSpreadSheetControl. This control is a GUI widget composed of a Grid control and a Tabbed Pane Control which display the values in KDCalc. Using the KDSpreadSheetControl, a SpreadSheet component can be added to any application.

Example:

```
KDSpreadSheetControl ssCtrl = new KDSpreadSheetControl();
this.add(ssCtrl);
sheetViewerCtrl.setCalcEngine(kdcEngine);
sheetViewerCtrl.setBounds(0,0,500,400);
```

The KDSpreadSheetControl can be integrated into most java Integrated Development Environments.

Testing with the KDCalc Engine for Java

KDSpreadSheet Viewer:

Included with KDCalc is an AWT based spreadsheet viewer. KDSpreadSheet is found in the KDCalcTools.jar in the kdyn.calc.tools package. The spreadsheet viewer makes it possible for users to view the .kdc file that was generated from the KDCalc Designer.

Within KDSpreadSheet, formulas cells are colored yellow, while cells with text or numbers are white. KDSpreadSheet does not support runtime editing of cell formulas. Formulas can only be changed by editing Excel and regenerating the .kdc file.

A batch file is included in the installation to run the KDSpreadSheet (*KDCalc\KDSpreadSheet.bat*).

CalcSpreadSheetBridge:

Included within the kdyn.calc.tools package is a class called CalcSpreadSheetBridge which allows you to communicate with both KDSpreadSheet and CalcEngine simultaneously. This debugging tool allows developers to simultaneously embed KDCalc into a running program and monitor the input/output in a grid format.

It is used like CalcEngine, but has a different accessor:

```
cEng = CalcSpreadSheetBridge.getKDSS(SS_NAME);
```

The usage of CalcSpreadSheetBridge is demonstrated in the TankGame sample source code.

Sample applications for Java

Three sample applications with Java source code are included. These will help familiarize you with KDCalc and will provide you with some reusable code. These samples are found in the *samples* directory installed with KDCalc.

HelloWorld

The HelloWorld applet/application is a very simple example to demonstrate the use of KDCalc. HelloWorld loads up the HelloWorld.kdc (generated from the HelloWorld.xls) and then gets and sets values from it.

Try compiling and running HelloWorld on your own using the compile.bat file or with a command like the following:

```
javac -classpath %CLASSPATH%;kdCalc.jar; kdyn\calc\demo\helloWorld\HelloWorldApp.java
```

Try making changes to the HelloWorld.xls and recreating the HelloWorld.kdc to see how it affects the application.

Tank Game

The Tank Game sample illustrates the use of KDCalc to provide calculations in a simple java game applet. A target is placed at a random height and distance away from a tank. The player has to set an angle and velocity for the tank to try to hit the target. After each shot a message and a score is displayed to the player. The player gets 10 shots per round and gets a total score for the round.

Before writing any code the game was designed in Excel, using cell formulas to calculate the trajectory of the tank shell. Cell formulas are also used to select the appropriate message to display to the player after each shot. Scores for each shot in the round are saved in cells, and the player's total score for the round is calculated in a cell.

After the behavior of the game was tested and approved, a user interface was painted and the code was written to pass the player's angle and velocity decisions into KDCalc. The calling applet then retrieves the calculated trajectory data from KDCalc, paints the tank shell motion path, and displays the calculated message to the player.

Internally, the Tank Game sample illustrates two methods of using KDCalc to calculate trajectories: continuous and discrete. Both methods are illustrated in the spreadsheet and in the calling applet.

In the discrete method, 10 cells calculate the position of the tank shell at 10 predefined points in the motion. In the continuous method, one cell is used to calculate the position of the shell at time T, so the calling applet can control the frequency of the calculations.

This is a very simple demo with only 2 user decision variables and a few outputs variables. In practice, KDCalc makes it possible to build simulations with dozens or hundreds of user decisions and millions of possible outcomes. See the 'Break Even Analysis' demo at www.KnowledgeDynamics.com for a more sophisticated example of a training simulation.

Investment Calculator

The Investment Calculator sample illustrates the use of KDCalc to provide projected return calculations in a DHTML page. The user sets a variety of investment decisions and sees the projected return on those investments over a user-determined number of years. The results are dynamically updated and displayed in both a graph and a table. This is a straightforward use of KDCalc in a very sophisticated DHTML page.

Performance Tester

The Performance Tester sample provides a calculation intensive workbook for performance and compliance testing. The workbook contains several worksheets that each test a particular type of Excel function. On each sheet, cell C5 is an input cell and cell C4 is the output cell. About 1000 cells must be recalculated to get the new value of C4 each time C5 changes.

Using this test spreadsheet, KDCalc Engine has proven to be up to 30 times faster than competing Java spreadsheet products, making it ideal for interactive applets and high-volume server applications.

SpreadSheetApplet

A simple example showing how to integrate the KDSpreadSheetControl into an Applet. Launch the index.htm within the example's directory to view the applet in action.

Feature Support Notes

(See *KDCalc Designer Users Guide.pdf* for details)

API Reference

(See attached file: *KDCalcDirjava*\docs\api\index.htm)**

Function Reference

(See attached file: *KDCalcDir*\KDCalc Functions.pdf)

Error Reference

Runtime errors:

(See attached javadocs)

End User License Agreement

(See attached)