



ExtenXLS™ Java | XLS reporting toolkit v.4

User's Manual and Programmer's Guide



Table of Contents

Introducing ExtenXLS	p 3
How to Use This Guide	p4
Quick-Start Guide	p5-6
Template Compatibility	p7
Performance Issues and Tuning	p8
Template Parsing Performance	p8
Conversion of Blank Records	p8
Jarloc and Licensekey Settings	p9
Runtime WorkBook Performance	p9
Debug Level	p9
String Performance Issues	p10
String Sharing	p11
How to use ExtenXLS	p13
The Application-Level Functions	
XML and HTML	p14
XML Report Definition Files	p15
Multiple Output Files from a Single Input Template Spreadsheet	16
Spreadsheet Output	16
The WorkBook Functions	17
Creating a New WorkBook	18
Deprecated WorkBookHandle Constructors	19
Modifying Existing WorkBooks	19
Copying and Inserting WorkSheets Between WorkBooks	20
Setting Rows and Column Size Default for WorkBooks	20
Creating, Accessing, and Modifying Named Ranges	21
The WorkSheet Functions	23
Creating, Copying, and Removing WorkSheets	23
Modifying Sheet Names	24
Getting, Setting and Reordering Tabs	24
Protect and Unprotect WorkSheets	25
Working with Rows and Columns	25
Adding and Removing Columns	28
Working with Charts	28
Grouping and Hiding Rows and Columns	30
Creating, accessing, and modifying CellRanges	31
Creating, Accessing, and Modifying NamedRanges	31
Cell Functions	33
Moving Cells	33
Removing Cells	34
Modifying Cell Formats	35
Modifying Cell Formats using the FormatHandle Object	35
Formatting Color, Style, and Text-Patterns	36
Adding Dates to Cells	38
Applying Font Attributes	39
Using Hyperlinks	40
Template Guidelines	41
Sample Applications	42
Additional Resources	42
Troubleshooting	43
A Note about ExtenBIS and the Extentech Roadmap	46

Introducing ExtenXLS 4

ExtenXLS is an easy-to-use reporting API that allows you to read, modify and create Excel* compatible spreadsheet-based reports from your Java applications.

Written entirely in Java, ExtenXLS creates new spreadsheet files from scratch as well as from existing spreadsheet files templates. ExtenXLS has the ability to utilize data from any source and output it in Excel*-compatible XLS, XML, and HTML formats.

Common uses for ExtenXLS range from a Java Servlet that produces financial analysis reports to Java beans embedded in JSP pages for use in executing formula calculations.

Today, companies around the world count on ExtenXLS in diverse mission-critical business applications. The ExtenXLS API can be used as a desktop spreadsheet component or embedded in a server application. With the ability to reuse existing spreadsheets, ExtenXLS can update template XLS files with fresh data from databases and output thousands of custom reports in seconds.

With its best-in-class Excel* compatibility, your template input files can range from a variety of complex legacy spreadsheets such as: surveys, checklists, what-if scenarios, and historical portfolio data.

With ExtenXLS, you're not limited to reports generated from existing template files. ExtenXLS has dozens of methods that can create new WorkBooks from scratch, allowing you to insert new Cells containing text, and numeric data. You can then customize the output with hundreds of fonts, colors, patterns, borders, formulas, named ranges, and built-in formats such as: currency, date, financial and numeric formatting patterns.

Other potential uses include: user interfaces built with the included Java swing spreadsheet components, and transformations of spreadsheets into HTML, and XML.

Since ExtenXLS is 100% Java, there is no need for native components or DLLs. And, like any good Java program, ExtenXLS ensures maximum portability of your applications no matter what the operating system platform.



How to Use This Guide

We hope that you find this guide informative and easy to use. The following conventions are applied throughout this manual:

Table 1.0 Code Conventions

Example Text	Reference API	Sample Code
Hyperlink Parameters	hlink_parameters	<pre><formula formula_cell="Invoice!G21" reference_location="Invoice!C21" reference_policy="0" /></pre>

Table 2.0 Icon Conventions

Icon	Reference
	The Tip icon is followed by examples or shortcuts to help you with your application.
	The Note icon gives you important information on variables that can occur in the application of your code.

Quick-Start Guide

This section is intended to get you started quickly programming with ExtenXLS.

ExtenXLS is a Java API designed for use by programmers familiar with the Java syntax, the concepts of file templates, and spreadsheet programs.

Basic use of the API requires the following steps:

1. Copy ExtenXLS.jar and extenxls.lic to a project directory. Make sure that the jar is in the classpath of your project. The ExtenXLS.jar file needs to be on your classpath in order for the JVM to find the ExtenXLS class files.
2. In your Java code, import the `com.extentech.ExtenXLS.*` package.
3. Create a new `WorkbookHandle`. This can either be a new, empty `Workbook` with the default of three sheets or you can provide an existing XLS file in the form of a byte array from a file or other data source. Review the `WorkbookHandle` constructors in the API documentation to see which `WorkbookHandle` is appropriate for your application.
4. Work with a `Worksheet` by using the **`WorksheetHandle`** **`WorkbookHandle.getWorksheet(String sheetname)`** method. You must catch the `WorksheetNotFoundException` in case the expected sheet does not exist already in the file.
5. Access Cell values using the **`CellHandle`** **`WorksheetHandle.getCell(String celladdress)`** method. You must catch the `CellNotFoundException` in case the expected Cell does not exist already in the file.



TIP: You can add a new Cell to the sheet and get a `CellHandle` with one line of code by using the **`CellHandle`** **`WorksheetHandle.add(Object ob, String address)`** method.

6. Set and get the value of Cells using the **`CellHandle.setCellVal(Object ob)`** and **`Object CellHandle.getCellVal()`** methods.

7. Stream the Workbook bytes to an output file using the **byte[] WorkbookHandle.getBytes()** method. If you are writing a web application, you can send the bytes over a ServletResponse to a web browser or you can write them out to a file or to any byte array consumer.

Template Compatibility

- ExtenXLS cannot modify or execute Macros. Unlike other Java reporting tools however, it does preserve existing VB code in templates and macros which can then be executed upon opening in your spreadsheet program.
- ExtenXLS does not allow for modification of certain features such as Cell notes, form objects or embedded images. These features will be preserved and retained however in the output files.
- For maximum compatibility, ExtenXLS uses the BIFF8 compatible format. Please ensure that your template files are version Office 97 or later before using them with the system. Do not save your files as "Excel 95/97" as this doubles the size of your output and creates a copy of the file in spreadsheet95, which ExtenXLS cannot modify.
- By design, ExtenXLS does not implement every XLS record type in the BIFF8 file format. Many of these settings are static in nature and/or can be determined by the template file instead of programmatically.

Performance Issues and Tuning

The memory and speed performance of ExtenXLS applications can vary dramatically based on a number of factors. The file size of input templates (especially those with many thousands of non-numeric or empty Cells) affects the speed of file parsing, writing, and memory usage.

ExtenXLS can be fine-tuned to enable the best mix of speed, compatibility (with external spreadsheet programs), and memory usage. When optimizing the performance of your application, care should be taken to minimize unnecessary input file content including large areas of empty Cells with formatting.

ExtenXLS performance issues and tuning can be broken down into two categories:

1. **Template Parsing Performance**—operations that affect only the reading or parsing of spreadsheet input files.
2. **Runtime Workbook Performance**—operations that are called after parsing the input file into a Workbook object.

Template Parsing Performance

To utilize the performance tuning and features of ExtenXLS that relate to the parsing of a spreadsheet file into a Java Workbook Object, you can set system properties to indicate how ExtenXLS should instantiate new WorkBooks.

Conversion of Blank Records

By default, Excel* converts blank records (Mulblanks) to formatted Cells. This conversion allows the setting of values on blank Cells without deleting/adding values. However, this conversion can significantly decrease the speed and performance of ExtenXLS when reading a template file. To enhance performance, you can tune ExtenXLS to change the internal Mulblank record type to a blank record type. To disable the conversion of blank Mulblank records on empty Cells set the System property "com.extentech.extenxls.convertmulblanks" to "false".

Example: `System.getProperties().put("com.extentech.extenxls.convertmulblanks", "false");`



Tip: If your input spreadsheets have many blank Cells (empty Cells with formatting) and you are experiencing performance issues reading your files, you

may wish to disable this feature and test your program to see if it is adversely affected.

Jarloc and Licensekey Settings

Due to issues on a number of Application Server Platforms, ExtenXLS may be unable to automatically find required resources. These include the location of the license file and default templates for new WorkbookHandles, and WorksheetHandles. These issues are easily resolved by setting the com.extentech.extenxls.licensekey and com.extentech.extenxls.jarloc system properties.

Licensekey Setting

Example:

```
String licensekey = "1255087426532DDD35268256220ABTRRW";  
System.setProperty("com.extentech.extenxls.licensekey", licensekey);
```

Jarloc Setting

Example:

```
String jarloc = "/ExtenXLS/ExtenXLSFilesNamedSomethingElse.jar";  
System.setProperty("com.extentech.extenxls.jarloc", jarloc);
```



Note: If you are running your application on an application server and are having difficulty instantiating WorkBooks, you should try using the server administration utility or other method to set the jarloc and licensekey properties for your server instances. This will override the automatic licensing mechanism.

Runtime Workbook Performance

There are a variety of set methods that can be called upon the Workbook to increase the performance tuning and features of ExtenXLS on an existing Workbook at runtime.

Debug Level

The internal workings of ExtenXLS can be examined for debugging purposes by setting the debuglevel value for the Workbook. The debuglevel outputs messages to System.out pertaining to the operation of the API, with levels of verbosity indicated by higher debuglevel values. The values range from 0 (no debug output, warnings and fatal error messages only) to 100 (very detailed information).

ExtenXLS User's Manual and Programmer's Guide

You can set the debug level using the **WorkbookHandle.setDebugLevel(int level)** on an existing Workbook.

Example:

```
WorkbookHandle.setDebugLevel(int level):  
mybook.setDebuglevel(10);
```

When debugging the parsing of a template, use one of the Workbook constructors containing a DebugLevel parameter.

Example: `mybook = new WorkbookHandle("C:/project/abook.xls", 10);`

String Performance Issues

Performance levels vary greatly on how character encoding is applied to Strings in the WorkbookHandle. Optimizing speed and performance depends on the needs of the application. Finding a tuning solution that is correct for your application is key to getting the most out of ExtenXLS. ExtenXLS by default uses UNICODE. Depending on the content of your Strings, this may result in larger files than Excel, but adding the Cells to the file will be considerably faster.

ExtenXLS can save space in output spreadsheet files if all characters in the String data are represented with a single byte (compressed). If a String contains characters that need two bytes to represent (such as eastern-language characters), then the String must be stored in an uncompressed Unicode format. You can set ExtenXLS to automatically detect the mode for each String or you can set to all Strings. The auto mode is the most flexible, but requires processing overhead and can be quite slow when adding many thousands of Cells. ExtenXLS has three modes for handling the internal encoding of String data added to a file.

If your files require mixed Unicode and non-Unicode Strings use:

WorkbookHandle.STRING_ENCODING_AUTO

Attributes:

-Slowest performance

-Smallest file size

ExtenXLS User's Manual and Programmer's Guide

If all Strings are in Unicode use:

WorkbookHandle.STRING_ENCODING_UNICODE

Attributes:

- Fastest performance
- Largest file size
- Safe for all Strings

If all the new Strings are non-Unicode with high byte compression ability use:

WorkbookHandle.STRING_ENCODING_COMPRESSED

Attributes:

- Faster than AUTO
- Smallest file size
- May corrupt double-byte Strings

String Sharing

String data in BIFF8 spreadsheets are stored in the shared String table. Depending on the requirements of your application, how you configure String sharing in WorkBooks can either be an important file size optimization or it be an unnecessary performance hit.

The Duplicate String Mode determines the behavior of the String table when inserting new Strings. By design, the String table shares a single entry for multiple Cells containing the same String. When multiple Cells have the same value, they share the same underlying String. Changing the value of any one of the Cells will change the value for any Cells sharing that reference. For this reason, it is important to determine how new duplicate Strings will be added to a spreadsheet. A duplicate String is one which already exists in the file. If subsequent changing of the values of these new Cells is required, you may inadvertently change the value of unintended Cells. To overcome this, set Duplicate String Mode to **ALLOWDUPES**. With this setting, if the String table

ExtenXLS User's Manual and Programmer's Guide

encounters a duplicate entry being added, it will insert a duplicate that can then be subsequently changed without affecting the other Cells sharing the original value.

Use the **Workbook.setDupeStringMode(int mode)** method to control the handling of new Strings to your Workbook.

Sharing identical Strings has a large impact on performance when the number of Strings gets too big. If an application is Sharing Strings, ExtenXLS must find the new String in the table in order to share it. If there is a large amount of String sharing in your application, this process can become very time-consuming.

Because of the performance and file size issues, you may wish NOT to share Strings, especially if you have very few duplicate Strings, and/or have very high performance requirements and a lot of Strings to add to the Workbook. Keep in mind that this will create a larger file size. Experimentation with the settings is encouraged to see what is optimal for your application.

To turn **OFF** String Sharing use **WorkbookHandle.ALLOWDUPES**

Example: `bookhandle.setDupeStringMode(WorkbookHandle.ALLOWDUPES);`

Attributes:

-Fastest performance

-Largest file size

To turn **ON** String sharing use **WorkbookHandle.SHAREDUPES**

Example: `bookhandle.setDupeStringMode(WorkbookHandle.SHAREDUPES);`

Attributes:

-Slowest performance

-Smallest file size

How to use ExtenXLS

ExtenXLS can be broken down into four-levels of functionality: Application, WorkBook, WorkSheet, and Cell levels. We will explore each of these levels in-depth to give you a greater understanding of how to fully utilize all that ExtenXLS has to offer.

Application-Level Functions

- Convert and output spreadsheets as XML, and HTML
- Use XML Report Definition Files to automate execution of database queries and mapping of results to spreadsheet templates
- Embed Java Swing spreadsheet components including formula (creation/execution), formatting, and named range support into GUI applications
- Generate multiple output files from a single input template spreadsheet merging different data into each output file
- Output spreadsheets to a variety of devices such as hard disks and networks, including client web browsers over HTTP, email clients, and XML consumers

WorkBook-Level Functions

- Create and modify new WorkBooks
- Calculate formulas
- Modify existing WorkBooks
- Copy and insert WorkSheets between WorkBooks
- Set row and column size defaults for WorkBooks
- Create, access, and modify named ranges

WorkSheet Functions

- Manipulate and copy charts
- Select and reorder sheet tabs
- Get and set header and footer text for printing WorkSheets
- Add, move, change, and delete rows and columns
- Protect and unprotect WorkSheets
- Get and set grouping and hiding for rows and columns
- Get and set row and column formats
- Get and set row and column size defaults for WorkSheets
- Get and set sheet names

ExtenXLS User's Manual and Programmer's Guide

- Add and remove Cells, rows, and columns

Cell Functions

- Modify Cell formatting
- Create and modify formulas
- Manipulate formula Cell references
- Calculate formulas and retrieve new values
- Add, move, modify and delete Cell values
- Create hyperlinks for Cells

The Application-Level Functions

The Application level functions cover the broad category of how ExtenXLS can be used in your program.

XML and HTML

New in ExtenXLS 4 is the ability to convert any spreadsheet to XML and HTML. With the XSLT file (included with your distribution), the API can transform the XML output to HTML using a single line of code.

This highly flexible architecture gives you the ability to create your own XSLT files with custom handling for application-specific output, as well as alternative output formats such as PDF.

You can also write XML applications that consume XML output from your spreadsheet reports. A template can be updated with data fetched from a database, results can be outputted as XML (formulas will be executed and the XML output will reflect the calculated values). The XML can then be used as an input to another application or process. This opens up a world of possibilities for reusing formula logic in spreadsheets as well as transforming report data for use in data warehousing applications.

XML Report Definition Files

Included in your distribution is the CellBinder API. CellBinder is a reporting framework that automates the creation of reports from a variety of data sources. The CellBinder API gives you the ability to map Resultset data into Cell ranges of a template spreadsheet

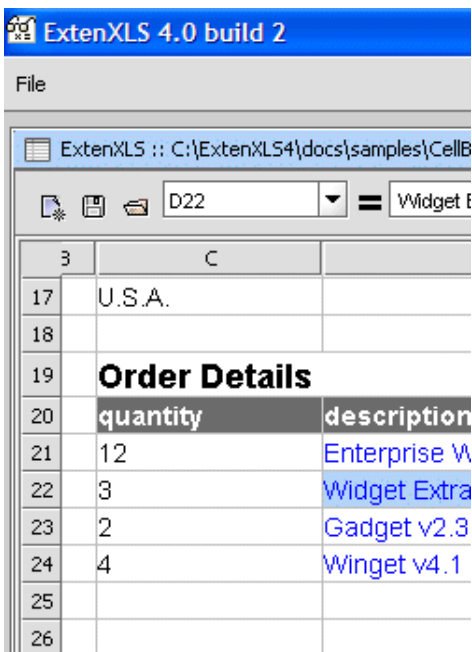
ExtenXLS User's Manual and Programmer's Guide

file. CellBinder also handles: the copying of existing formats into spreadsheets, copying and updating of formulas, and the updating of charts with new rows of data. CellBinder also can apply dynamic hyperlinks on Cells, allowing you to create 'drill-down' reports that when clicked, pass in row-specific data to a URL.

Embedded Java Swing spreadsheet Components

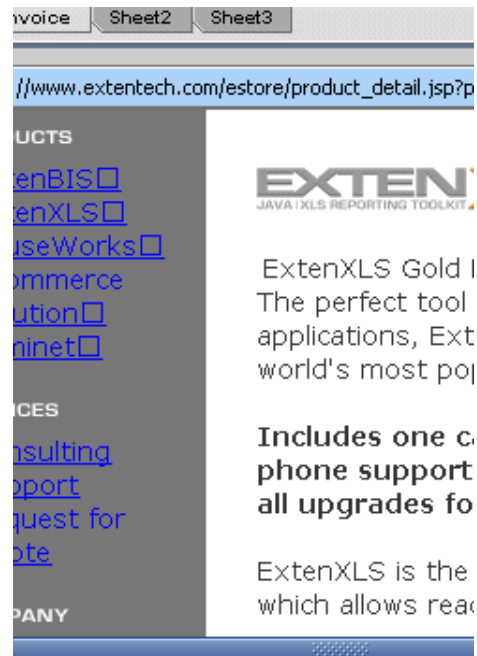
Writing your own spreadsheet component can be very expensive. The ability to embed a Java spreadsheet in your application with a few lines of code can save countless hours of development. The all-Java Swing spreadsheet components in ExtenXLS provide drop-in spreadsheet functionality with the ability to create and execute formulas, edit data in Cells, open and save Excel-compatible spreadsheets, open and browse URLs from hyperlinked Cells, and more.

Spreadsheet with hyperperlinks



The screenshot shows the ExtenXLS 4.0 build 2 application window. The spreadsheet has a menu bar with 'File', a toolbar with icons for file operations, and a formula bar showing 'D22 = Widget t'. The spreadsheet content includes a table with the following data:

	3	C	
17		U.S.A.	
18			
19		Order Details	
20		quantity	description
21		12	Enterprise W
22		3	Widget Extra
23		2	Gadget v2.3
24		4	Winget v4.1
25			
26			



Multiple Output Files from a Single Input Template Spreadsheet

The best way to get the most out of existing investments in spreadsheet development is by updating existing spreadsheets with data from databases. The ability to reuse

ExtenXLS User's Manual and Programmer's Guide

formulas, charts, formatting and VB macros from a single template and output it many thousands of times with different data saves countless hours of copying and pasting, eliminating human error, and ensuring standardization within your organization.

Whether you are outputting financial reports, compliance checklists, or portfolio performance charts, using ExtenXLS will ensure that your end-users are able to access their information in a highly functional format that they are confidently familiar with.

Spreadsheet Output

ExtenXLS can be output to a variety of devices such as: XML, HTML, networks, including client web browsers, over HTTP, email clients, and hard disks. By automating the generation of reports, users get their information when and where it is most useful to them.

The Workbook Functions

ExtenXLS is not solely a template-based API. With ExtenXLS, you can create new spreadsheet files, Cells, formats, and formulas. Using the dynamic capabilities of ExtenXLS you can also convert data programmatically into spreadsheet files, and then handle the spreadsheet output as you would any other application.

The `com.extentech.ExtenXLS.WorkBookHandle` object is your primary handle to your spreadsheet file. The `WorkBookHandle` object represents the `WorkBook` portion of the spreadsheet file (there are other portions of the file which ExtenXLS preserves but does not modify, such as those containing VB code). The `WorkBookHandle` class provides convenient access methods for the various functions of spreadsheet objects.

If you are reading from an existing template, and would like to output to that file, you must create a file input and output file path.

Example input path: `String finpath = "explore.xls";`

Example output path: `String foutpath = "explored.xls";`

Next, read in the `WorkBook` from your template spreadsheet file.

Example:

```
WorkBookHandle book = new WorkBookHandle("Desktop:Java:eclipse.1:workspace:ExtenXLS:explore.xls");  
  
System.out.println("ExtenXLS version " + book.getVersion() + " loaded.");
```

Then Get a handle to the `WorkSheet` you want to work with

Example: `WorkSheetHandle sheet1 = book.getWorkSheet("Sheet1");`

Creating a New Workbook

Instantiating a `WorkbookHandle` class parses the binary spreadsheet file data (BIFF8 format) and creates Java objects that represent the document's Cells, values, formatting, and settings. The following requirements are needed before instantiating a `WorkbookHandle`:

- Java VM version 1.4 or higher
- A valid `ExtenXLS.jar` file in your classpath
- A valid license key for your installation in the same directory as the jar



Note: if your application is deployed within an EAR, WAR file, or any other case where the classes within the `ExtenXLS.jar` are repackaged into another file, you will need to specify the location of the file containing the `ExtenXLS` class files by setting the system property to `"com.extentech.extenxls.jarloc"` property. Additionally, you will need to set the license key for `ExtenXLS` using `"com.extentech.extenxls.licensekey"` system property.

The constructor of the `WorkbookHandle` class reads from an array of file bytes. The data can be supplied from a database or network stream. You can create a `WorkbookHandle` on a new, empty `Workbook` with the empty constructor. The empty constructor provides a default of three `Worksheets` per `Workbook` (`Sheet1`, `Sheet2`, and `Sheet3`).

To create a new `Workbook` use the **`WorkbookHandle()`** constructor.

Example: `WorkbookHandle book = new Workbook();`

To read from an existing `Workbook` use **`WorkbookHandle(String)`**

Example (OSX): `WorkbookHandle book = new WorkbookHandle("Desktop:ExtenXLS4:explore.xls");`

Example (Unix): `WorkbookHandle book = new WorkbookHandle("/home/username/ExtenXLS4/explore.xls");`

Example (Windows): `WorkbookHandle book = new WorkbookHandle("C:/ExtenXLS4/explore.xls");`



Tip: There are a variety of `WorkbookHandle` constructors that allow you to create `WorkbookHandles` from byte arrays and files, as well as giving you the ability to output debug information on the internal operation of the API as it parses your input file. Please review the API docs included with your distribution for details on all available `WorkbookHandle` constructors.

Deprecated `WorkbookHandle` Constructors

In order to lessen the complexity of the API, we have removed three `WorkbookHandle` constructors, which take a 'Working Directory' String parameter to denote the location of the template file.

Template path Strings can be constructed prior to calling the simplified constructors, which now take a fully qualified path to the template file.

The removed constructors are:

`public WorkbookHandle(String wd, String fname)`

`public WorkbookHandle(String wd, String fname, boolean relative)`

`public WorkbookHandle(byte[] barray, String workingdir)`

For additional information on changes to the API from prior versions, please see the "ExtenXLS_Upgrade_Guide.pdf" in your docs/manual folder.

Modifying Existing WorkBooks

One of the many functions of ExtenXLS is in the use of reusability of existing files. This is an excellent way to provide reports with interactive features using VB macro code. To reuse spreadsheet templates files in this way, read in the `Workbook` containing VB macros and update the relevant values from a database (or any other source), then output the file as a new `Workbook`.

When the output file is opened in Excel, the VB macros will execute, referencing the updated values. This functionality preserves feature-rich reports by retaining all the interactivity your users are accustomed to.

Another popular way to use ExtenXLS is to populate `WorkBooks` with a variety of template `Worksheets`. To do this, you need to create a master `Workbook` containing all

ExtenXLS User's Manual and Programmer's Guide

of the WorkSheets. After formatting and developing your master Workbook containing all possible WorkSheets, create an empty or sheetless version of the original Workbook. Next, populate this empty master Workbook with the appropriate WorkSheets by re-adding in any of the WorkSheets from the original master Workbook as necessary.

ExtenXLS provides two helper methods to assist with this scenario:

To get an empty duplicate of the Master that contains all of the shared formatting information, use the **getNoSheetWorkbook** method

Example: `WorkbookHandle bookRemoved = masterBook.getNoSheetWorkbook();`

To pick a sheet from the master Workbook, use the **WorksheetHandle WorkbookHandle.getWorksheet(String sheetname)** method.

Example: `bookRemoved.addSheetFromWorkbook(masterBook,"SourceSheet","NewSheetName");`

In the example listed above, the "bookRemoved" Workbook will contain a copy of the Worksheet "SourceSheet" from the original, master Workbook. The new sheet is named "NewSheetName". Formatting will be retained from the original Workbook. The new Workbook can be further manipulated and worked with using ExtenXLS.

Copying and Inserting Worksheets Between WorkBooks

The WorksheetHandle allows access to a particular Worksheet within your WorkbookHandle. In order to create this handle, use the `getWorksheet(String sheetname)` method within the WorkbookHandle.

Example: `WorksheetHandle sheet = book.getWorksheet("Sheet1");`

You can get an array of handles to all of the Worksheets in the Workbook using the **WorksheetHandle[] WorkbookHandle.getWorksheets()** method.

Example: `WorksheetHandle[] sheets = book.getWorksheets();`

Setting Rows and Column Size Default for WorkBooks

ExtenXLS User's Manual and Programmer's Guide

WorkBook row and column sizes can be set and applied whenever a specific row or column size is unspecified. The RowHandle is used to work with individual rows in an XLS file. To set the default height for all rows use the

WorkbookHandle.setDefaultRowHeight(int x) method.

Example: mybook.setDefaultRowHeight(10000);

The ColHandle is used to work with individual Columns in an XLS file. To set the default formatting for all columns use the WorkbookHandle.setDefaultColWidth(int x) method.

Example: mybook.setDefaultColWidth(10000);

Creating, Accessing, and Modifying Named Ranges

To access an existing named range in a sheet, use **NameHandle** **getNamedRange(String name)** to retrieve the range.

Example: NameHandle nand = mybook.getNamedRange("nametest4");

Then use the **getCells()** method to access all the Cells in the range

Example:

```
CellHandle[] ch = nand.getCells(); // access the Cells in the range
for(int x = 0;x<ch.length;x++){
    ch[x].setVal(123 * x);
    ch[x].setFontColor(FormatHandle.PaleBlue);
}
System.out.println(nand.getName());
```

To create a range of Cells, use the **CellRange(String range, WorkbookHandle bk)**. Next, use the **NameHandle(String name, CellRange range)** method to generate a named range consisting of the range of Cells specified.

Example: CellRange range = new CellRange("Sheet1!D8:D13", mybook);
NameHandle newname = new NameHandle("NewNamedRange",range);

Calculating Formulas

To provide improved performance, as well as a finer-grained control over formula calculations, ExtenXLS allows you to turn off the automatic calculation of formulas by using the **setFormulaCalculationMode()** method.

Example: `book.setFormulaCalculationMode(WorkBookHandle.CALCULATE_EXPLICIT);`

The WorkSheet Functions

The WorkSheetHandle provides a handle to a WorkSheet within an XLS file and includes convenience methods for working with the Cell values within a sheet. The WorkSheetHandle allows access to a particular WorkSheet within your WorkBook. In order to gain access to this handle, use the `getWorkSheet(String sheetName)` method within the WorkBook object.

Example:

```
WorkBook mybook = new WorkBook();
WorkSheetHandle sheet = mybook.getWorkSheet(Sheet1);
```

You can get an array of handles to all of the WorkSheets in the WorkBook using the **getWorkSheets()** method:

Example:

```
WorkSheetHandle[] sheets = book.getWorkSheets();
```

This will return an array of handles to all of the WorkSheets in the WorkBook allowing you the flexibility to work on each one individually.

Creating, Copying, and Removing WorkSheets

To create a new WorkSheet with the **createWorkSheet(String newSheet)** method. This will insert a new WorkSheet and place it at the end of the WorkBook.

Example:

```
WorkSheetHandle mysheet = mybook.createWorkSheet("newsheet");
```

To delete any number of WorkSheets, use the remove method. For your convenience, you can remove all WorkSheets in a WorkBook by using the **removeAllWorkSheets()** method.

Example:

```
mybook.remove("mysheet");
```

Example:

```
mybook.removeAllWorkSheets();
```

ExtenXLS User's Manual and Programmer's Guide

To copy an existing WorkSheet to your WorkBook, use the **copyWorkSheet(String SourceSheetName NewSheetName)** method. This will duplicate a WorkSheet in the WorkBook and add it to the end of the WorkBook with a new name.

Example:

```
WorkBook thisbook = new WorkBook();
WorkSheetHandle sheet = thisbook.getWorkSheet("Sheet1");
thisbook.createWorkSheet("NEW SHEET1");
thisbook.copyWorkSheet("Sheet1", "NEW SHEET2");
```

Modifying Sheet Names

If you wish to rename a WorkSheet, use the **WorkSheetHandle.setSheetName(String newName)** method. This method will change the name on the WorkSheet's tab as well as all programmatic and internal references to the name.

Example: `mysheet.setSheetName("Sheet1", "Records");`

To get the name of a WorkSheet use the **getSheetName()** method.

Example: `String sheetname = mysheet.getSheetName();`

Getting, Setting and Reordering Tabs

Sheets can be selected so that they appear as the first visible sheet when the output file is opened. You can also reorder sheets to your specifications. To select a WorkSheet and it to the first visible tab in the WorkBook use the **setFirstVisibleTab(int x)** method.

Example: `WorkSheetHandle WorkSheet4 = new WorkSheet();`
`WorkSheet4.setTabIndex(0);`

To reorder the display of the WorkSheet tabs use the **setTabIndex(int idx)** method. This is a zero-based index. Thus, the first tab you will see displayed in your WorkBook will be 0.

Example: `mysheet.setTabIndex(5);`

Selecting Header and Footer Text for Printing WorkSheets

To change or set the header text, use the **setHeaderText(String headerTextName)** method. This will change the header to the desired text.

Example: `setHeaderText("Sales Report For Acme Corp");`

To set or change the footer text use, the **setFooterText(String footerTextName)** method.

Example: `setFooterText("Exploring ExtenXLS is Fun!");`

Protect and Unprotect WorkSheets

WorkSheet protection can be set or removed to control any modifications in the output file. To lock the values of a WorkSheet, set the WorkSheet to protect by using the **setProtected(boolean b)** method.

Example: `mysheet.setProtected(true);`

To unlock the protected WorkSheets, use the **setProtected(boolean b)** method.

Example: `mysheet.setProtected(false);`

Working with Rows and Columns

The RowHandle is used to work with individual rows in an XLS file. You can use the RowHandle to:

- Set the default formatting for a row
- Change the height of a row
- Get an array of the Cells in a row
- Hide or collapse a row

To get a handle to an array Cells in a row use the get **getCells()** method.

Example:

```
try{
    RowHandle row1 = sheet1.getRow(0);
    CellHandle[] cells = row1.getCells();
}catch(RowNotFoundException a) {
    // boo!
}
```

ExtenXLS User's Manual and Programmer's Guide

Rows can be sized for the entire sheet, and will be used unless otherwise specified. To change the height of a row, use the **setHeight(int i) method**.

Example:

```
RowHandle row1 = sheet1.getRow(0);
row1.setHeight(10000);
```

To hide a row, use the **setHidden(boolean b) method**.

Example:

```
RowHandle row1 = sheet1.getRow(0);
row1.setHidden(true);
```

To collapse a row, use the **setCollapsed(boolean b) method**.

Example:

```
row1.setCollapsed(true);
```

To set the default formatting for a row use the **setFormatId(int i) method**. This will set the default for all the Cells in the specified row.

Example:

```
row1.setFormatId(myFormattedCell.getFormatId());
```

The ColHandle object provides access to columns and its Cells within a WorkSheet. You can use the ColHandle to work with individual columns in an XLS file. With a ColHandle you can:

- Get a handle to the Cells in a column
- Set the width of a column
- Hide or collapse a column
- Set the default formatting for a column

To get a handle to an array of Cells within a column use the **getCells()** method.

Example:

```
ColHandle colA = sheet1.getCol(0);
CellHandle[] cells = colA.getCells();
```

ExtenXLS User's Manual and Programmer's Guide

Columns widths can be specified for an entire sheet. To set the width of a column, use the **setWidth(int i)** method.

Example: `colA.setWidth(10000);`

To hide a column, use the **setHidden(boolean b)** method.

Example: `colA.setHidden(true);`

To collapse a column, use the **setCollapsed(boolean b)** method.

Example: `colA.setCollapsed(true);`

To set the default formatting for a column, use the **setFormatId(int i)** method. This will set the default for all the Cells in the specified column.

Example: `colA.setFormatId(myFormattedCell.getFormatId());`

Adding and Removing Rows

To insert a new row into a WorkSheet use the **insertRow(int rownum)** method. This will insert a blank row into a WorkSheet and shift all rows below the Cell down one.

Example: `sheet1.insertRow(12);`



Note: This method is only necessary to move existing Cells by inserting empty rows.



Tip: If you would like to create a new row that retains all the formatting of the prior row, use the **insertRow (int rownum,boolean shiftrows)** method. This will shift all existing subsequent rows down by one.

Example: `sheet1.insertRow(3,true);`

To remove a row from a WorkSheet, use the **removeRow(int)** method. This will remove the row and all associated Cells from a WorkSheet.

Example: `mysheet.removeRow(10);`

ExtenXLS User's Manual and Programmer's Guide

To remove a row and shift all rows below the target up one, use the **removeRow(int rownum boolean shiftrows)** method.

Example: `sheet1.removeRow(3,true);`

Adding and Removing Columns

To insert a column into a WorkSheet, use the **insertCol (String colNum)** method. This will insert a blank column into the WorkSheet and shift all columns to the right of the Cell over one. Adding new Cells to non-existent columns will automatically create new columns in the file.

Example: `mysheet.insertCol("H");`



Note: This method is only necessary to move existing Cells by inserting empty columns

To remove a column and all associated Cells within a WorkSheet use the **removeCol(String colstr)** method.

Example: `sheet1.removeCol("C");`

To remove a column and shift all columns to the right of the target column by one, use the **removeCol(String colstr boolean shiftcols)** method.

Example: `sheet1.removeCol("C", false);`

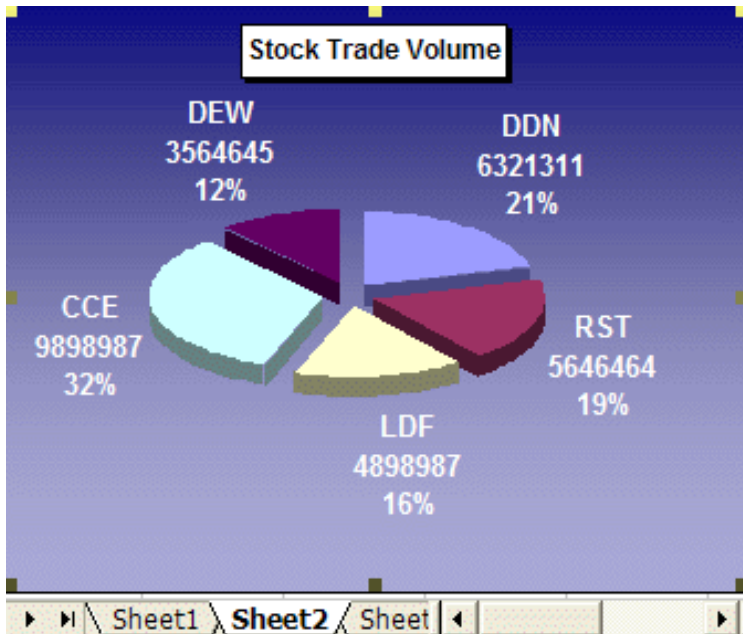
Working with Charts

With a ChartHandle, you can add Cells to a chart, change a chart's title, axis label, and copy charts to another WorkSheet or WorkBook. Begin by retrieving a ChartHandle from an existing chart in a WorkSheet. To retrieve a ChartHandle from an existing chart in a WorkSheet, use the **ChartHandle WorkSheetHandle.getChart(String chartname)** method.

Example:

```
try{
    ChartHandle performancechart = mysheet.getChart("Performance");
}catch(ChartNotFoundException e){
    // oops!
}
```

Chart output modified by ExtenXLS 1



Using a ChartHandle, you can add Cells to the chart using the boolean **ChartHandle.changeSeriesRange(String originalrange, String newrange)** method.

Example:

```
if(performancechart.changeSeriesRange("Sheet1!C23:E23","Sheet1!C23:G23"));
    System.out.println("Successfully Added Columns F and G to Series Range");
```

To change the chart title, use the **Charthandle.setTitle(String title)** method.

Example: mychart.setTitle("NEW CHART!");

To change the axis labels, or other text labels in the chart use the **ChartHandle.changeTextValue(String originaltext, String newtext)** method.

ExtenXLS User's Manual and Programmer's Guide

Example:

```
if(ct.changeTextValue("50 Meter Dash Times", "High Jump Distance"))
    System.out.println("Successfully Changed Categories Label");
```

To copy a chart into another sheet or Workbook, use the **copychartToSheet(String Chart name, String sheet name)** method.

Example:

```
try{
    book.copyChartToSheet("New Chart Title", "Sheet3");
}catch(Exception e){
    System.out.println(e);
}
```

Grouping and Hiding Rows and Columns

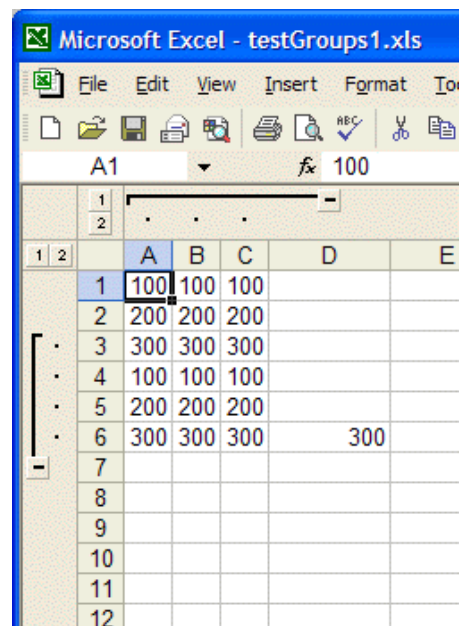
Grouping rows and columns is a great presentation tool that allows for the hiding of large quantities of detailed information into section summaries. When more detail is required, the hidden information can be expanded to display all the desired information.

To group row and columns, use the get/set methods for the collapsed, outlinelevel, and hidden methods in both the RowHandle and ColHandle classes. *See page 25 in this guide or visit the API included with in your distribution for more detailed information on these methods.*

Collapsed: the state for the row/col determining whether the '+' or '-' sign is visible in the spreadsheet program.

Outlinelevel: row/column groups can be nested. The outlinelevel determines which level of grouping the row/column is assigned.

Hidden: whether the row/column is visible. Collapsed rows and columns are also not visible.



Creating, Accessing, and Modifying CellRanges

CellRanges provide convenient methods for working with logical groups of Cells. The following is a list of CellRange constructors.

This constructor will create missing Cells within the range automatically:

Example: `CellRange databaseValues = new CellRange("Sheet1!A1:B8", mybook);`

This constructor will not create missing Cells within the range automatically:

Example: `CellRange databaseValues = new CellRange("Sheet1!A1:B8", mybook, false);`

This constructor will create a 3D range of Cells, spanning WorkSheets:

Example: `CellRange databaseValues = new CellRange("Sheet1!A1:Sheet3!B8", mybook);`

You can add new Cells to a range. This will expand the rectangle of Cells referenced by the CellRange to include the added Cell.

Example:

```
CellRange cellz = new CellRange("Sheet1!A1:A20", mybook);
CellHandle b1 = sheet.getCell("B1");
cellz.addCellToRange(b1);
CellHandle[] rangeCells = cellz.getCells(); // now contains A1:B20
```

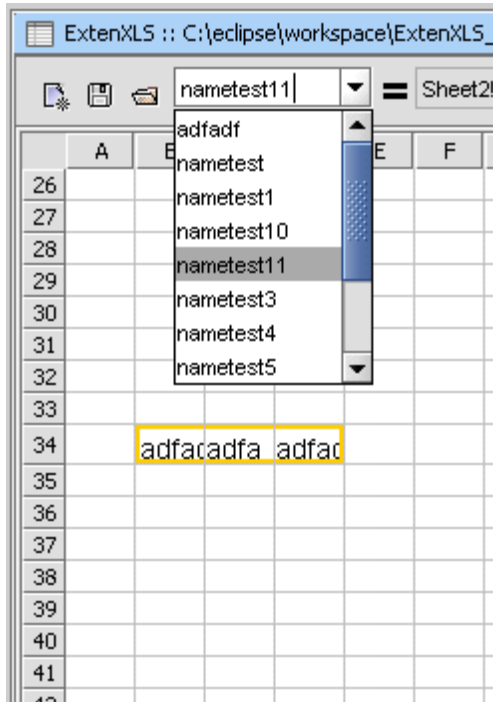
Creating, Accessing, and Modifying NamedRanges

Named ranges can be created with a single line of code. Cells can be added (expanding the named range) or deleted (shrinking the named range.) Also, named ranges can be defined as three-dimensional and span multiple sheets.

Named ranges can be created from a CellRange with a single line of code:

Example: `NameHandle valueCells = new NameHandle("ValueCells",mycellrange);`

Example of Named Ranges in ExtenXLS



The main difference between NamedRanges and CellRanges is that CellRanges are for convenience within the ExtenXLS API, whereas NamedRanges are a function of the spreadsheet. It will display and be accessible from the Named Range dropdown box in your spreadsheet program.

Cell Functions

ExtenXLS makes it convenient for you to work with Cells individually. The Cell functions allow you to modify and apply dynamic formatting to the Cells in your spreadsheet.

Some of the Cell attributes that you can edit or apply are:

- Moving Cells
- Removing Cells
- Getting and setting formatting patterns
- Getting and setting font styles
- Getting and setting border styles and colors
- Getting and setting background and foreground patterns and colors
- Add Dates
- Add hyperlinks to Cells
- Change the value of existing Cells

Moving Cells

The CellHandle provides a handle to an XLS Cell and its values. Use the CellHandle to work with individual Cells in an XLS file. Cell objects are created through the parsing process and are the atomic element which spreadsheet table Cells can be addressed. Using the CellHandle class, new values can be set on existing Cells based on any data source available to your Java program.

To get a handle to a particular CellHandle in the WorkSheet, use the **CellHandle WorkSheetHandle.getCell(String address)** method and pass in a String representing the address of the Cell in the WorkSheet:

Example: CellHandle Cell = sheet.getCell("B2");

To obtain the value of a Cell from a WorkSheet:

Example: String str = Cell.getStringVal();

This will get the String representation of the Cell value

ExtenXLS User's Manual and Programmer's Guide

Or use:

Example: Object obj = Cell.getVal();

This will get the actual Cell value (Double, String, Integer, etc.).

To change the value of the Cell, simply use the **setVal(String val)** method

Example: mycell.setVal("new value for Cell");

To instantiate a CellHandle object, you must first have a valid WorkSheet and WorkbookHandle object.



Note: You can add a Cell or a group of Cells to your WorkSheet by using the WorkSheetHandle object. *For more information on the WorkSheetHandle object, Please see the "Working with Rows and Columns" on page 25 of this manual*

To move a Cell to a different column, use the **CellHandle.moveTo(String newaddress)** method.

Example: mycell.moveTo ("Sheet1:B2");

To move a Cell to a different column, use the **CellHandle.moveToCol(String newcol)** method.

Example: mycell.moveToCol("B");

To move a Cell to different row, use the **CellHandle.moveToRow(int newrow)** method

Example: mycell.moveToRow(10);



Note: If there is an existing Cell in the specified address CellHandle will throw a CellPositionConflictException object.

Removing Cells

To remove a Cell from your WorkSheet, use the **remove(boolean nullme)** method.

Example: mycell.remove(true);

Modifying Cell Formats

There are three basic approaches to working with Cell formats using ExtenXLS:

1. Modify the attributes of a Cell's format using the CellHandle object.
2. Create a new FormatHandle object, modify its attributes, and apply it to Cells within the Workbook. This is recommended in most cases.
3. Put Cells with your desired format in your template input file, then apply their format to other Cells dynamically using **targetCell.setFormatId(sourceCell.getFormatId())** methods.

The FormatHandle object gives you the flexibility to apply formatting to individual and multiple Cells. You can also apply formats to Cells by using the CellHandle object. In most cases, we recommend using the FormatHandle. Instantiating a FormatHandle object will create a new unshared format object, which can subsequently be applied to (and shared by) only the Cells you intend.

Modifying Cell Formats using the FormatHandle Object

Like many of the objects in the spreadsheet Workbook, format objects are stored in the file as shared records, which can be applied to many different Cells in the Workbook.

In every new spreadsheet file there is a default format within the file that is applied to all new Cells and existing Cells that have not had any explicit format settings applied to them. To share any Cell's format information using ExtenXLS, you should use the: **getFormatId()** method to get a pointer to this format, and then call: **setFormatId(int x)** to apply the shared format to another Cell.

The FormatHandle object gives you more control over exactly which Cells will acquire new formats – which is an advantage over setting formatting values on CellHandles that may change other Cells sharing the underlying format object.



Warning: When you change the Cell formatting using the attributes of a CellHandle those changes will be reflected in all Cells sharing this format.

You may wish to change the default format for every unformatted Cell in your file, in which case, grabbing any unformatted Cell and modifying its format attributes will alter

ExtenXLS User's Manual and Programmer's Guide

the underlying format for all unformatted Cells, thus giving your file a new default' format.




























Formatting Color, Style, and Text-Patterns

The FormatHandle Class contains numeric (int) constants that supply the information on which colors and patterns to apply to the format. To access these constants, use the

FormatHandle.CONSTANTNAME.

Example:
`mycell.setCellBackgroundColor(FormatHandle.COLOR_LT_BLUE);`
`mycell.setCellForegroundColor(FormatHandle.COLOR_GOLD);`
`mycell.setCellBackgroundPattern(FormatHandle.PATTERN_HOR_STRIPES);`

Color Formatting Constants in FormatHandle:

COLOR_BLACK		COLOR_BURGUNDY		32		COLOR_BLUE1		
COLOR_WHITE		COLOR_DK_GREEN		33		COLOR_BLUE2		
COLOR_RED		COLOR_PURPLE		34		COLOR_LIME_GREEN		
COLOR_GREEN		COLOR_BROWN		35		COLOR_GOLD		
COLOR_BLUE		COLOR_DK_PINK		36		COLOR_LT_ORANGE		
COLOR_YELLOW		COLOR_TEAL2		37		COLOR_ORANGE		
COLOR_MAGENTA		COLOR_GRAY1		38		COLOR_BLUE_GRAY		
COLOR_TEAL1		COLOR_GRAY		39		COLOR_GRAY4		
8		COLOR_BABY_BLUE		40		COLOR_AQUA		
9		COLOR_MAGENTA2		41		COLOR_GRAY5		
10		COLOR_OFFWHITE		42		COLOR_DK_BROWN		
11		COLOR_GRAY2			COLOR_TAN1		COLOR_MED_BROWN	
12		COLOR_NAVY_BLUE			COLOR_TEAL3		COLOR_BROWN2	
13		COLOR_PINK			COLOR_PINK3		COLOR_DK_PINK2	
14		COLOR_LT_BLUE			COLOR_MAUVE		COLOR_DK_BLUE	
15		COLOR_GRAY3			COLOR_TAN2		COLOR_DK_GRAY	

ExtenXLS User's Manual and Programmer's Guide

In addition to color and style formatting, text-pattern based Cell formats can also be applied to your Cells using the following patterns

Pattern Formatting Constants in FormatHandle:

PATTERN_NONE	Text	PATTERN_CROSSPATCH1	Text
PATTERN_FILLED	Text	PATTERN_MED_DOTS	Text
PATTERN_LIGHT_FILL	Text	PATTERN_LIGHT_DOTS	Text
PATTERN_MED_FILL	Text	PATTERN_HVY_DOTS	Text
PATTERN_HVY_FILL	Text	PATTERN_DIAG_STRIPES5	Text
PATTERN_HOR_STRIPES1	Text	PATTERN_DIAG_STRIPES6	Text
PATTERN_VERT_STRIPES1	Text	PATTERN_DIAG_STRIPES7	Text
PATTERN_DIAG_STRIPES1	Text	PATTERN_DIAG_STRIPES8	Text
PATTERN_DIAG_STRIPES2	Text	PATTERN_VERT_STRIPES3	Text
PATTERN_CHECKERBOARD1	Text	PATTERN_HOR_STRIPES3	Text
PATTERN_CHECKERBOARD2	Text	PATTERN_VERT_STRIPES4	Text
PATTERN_HOR_STRIPES2	Text	PATTERN_HOR_STRIPES4	Text
PATTERN_VERT_STRIPES2	Text	PATTERN_PATCHY1	Text
PATTERN_DIAG_STRIPES3	Text	PATTERN_PATCHY2	Text
PATTERN_DIAG_STRIPES4	Text	PATTERN_PATCHY3	Text
PATTERN_GRID1	Text	PATTERN_PATCHY4	Text

Currency Constants in FormatHandle:

"General"

"Currency"

"0"

"0.00"

"#,##0"

"#,##0.00"

"(\$#,##0);(\$#,##0)"

"(\$#,##0);[Red](\$#,##0)"

"(\$#,##0.00);[Red](\$#,##0.00)"

"(\$#,##0.00);[Red](\$#,##0.00)"

"(##0_);(##0)"

"(##0_);[Red](\$#,##0)"

"(##0.00_);[Red](\$#,##0.00)"

"(##0.00_);[Red](\$#,##0.00)"

ExtenXLS User's Manual and Programmer's Guide

```
"_(*#,##0_);_(*($#,##0);_(*\"-\"_);_(@_)"
"_$*#,##0_);_($*($#,##0);_($*\"-\"_);_(@_)"
"_*#,##0_);_(*($#,##0);_(*\"-\"??_);_(@_)"
"_$*#,##0_);_($*($#,##0);_($*\"-\"??_);_(@_)"
"0%"
"0.00%"
"0.00E+00"
"# ?/?"
"# ??/??"
```

Adding Dates to Cells

ExtenXLS gives you the ability to apply date formatting to Cells. You can add dates directly to a WorkSheet or apply date formatting to individual Cells.

To add dates directly to your WorkSheet use the

WorkSheetHandle.add(java.util.Date dt, String address) method.

```
Example: java.util.Date d = new java.util.Date(System.currentTimeMillis());
mysheet.add(d,"A1");
```

You can also specify a date Format by using the **WorkSheetHandle.add(java.util.Date dt, String address, String format)** method.

```
Example: java.util.Date d = new java.util.Date(System.currentTimeMillis());
mysheet.add(d,"A1", "mm/dd/yy");
```

Entering a null parameter for the format specifies the default date format (m/d/yy h:mm).

There are a host of date formats accessible through the `com.extentech.ExtenXLS.DataConverter` helper class.

ExtenXLS User's Manual and Programmer's Guide

Listed in the table below are the ExtenXLS valid date format patterns:

ExtenXLS Date Format Patterns

m/d/y	h:mm
d-mmm-yy	h:mm:ss
d-mmm	m/d/yy h:mm
mmm-yy	mm:ss
h:mm AM/PM	[h]:mm:ss
h:mm:ss AM/PM	mm:ss.0

Applying Font Attributes

To modify a font for a CellHandle, use the **setFont(String fontname, int fontstyle, int fontsize)** method of the CellHandle. This will create a new format definition in the file, and will not be shared by other Cells with the original shared format.

Below is an example of efficient use of the CellHandle formatting methods within a new Workbook:

```
void testFormats(String finpath, String sheetname){
    WorkbookHandle tbo = new WorkbookHandle();
    WorksheetHandle sheet1 = tbo.getWorksheet(sheetname);
    try{
        sheet1.add("Eurostile Template Cell","A1");
        catch(CellPositionConflictException e){System.out.println(e);}
        int SHAREDFORMAT = 0;
        CellHandle b = null;
        CellHandle a= null;
        try{
            b = sheet1.getCell("A1");
            // Create a new Font format
            b.setFont("Eurostile",Font.BOLD,14); // using setFont() creates a NEW Format Record – not shared!
            for(int t = 1; t<=10;t++){
                try{
                    sheet1.add(new Float(t*67.5),"E" + t);
                    catch(CellPositionConflictException e){System.out.println(e);}
                    a = sheet1.getCell("E" + t);
                    // Share the format created above
                    a.setFormatId(SHAREDFORMAT); // share the new format explicitly
                }
            }
            a.setFont("Tango",Font.BOLD,16);
            // set a format pattern
            a.setFormatPattern("[h]:mm:ss");
            a.getCol().setWidth(5000);

            t(sheetname, sheetname + " Copy");
        }
    }
}
```

```
cell1 = sheet1.getCell(addr);
cell1.setFormat(fmt1);
```



Warning: If you use **setFont(name,type,size)** on many Cells, the output file size can increase unnecessarily (depending on the number of Cells you are working with).

For this reason, we recommend the use of a shared FormatHandle for each format you wish to utilize, then adding cells to it that you want to apply the format to.

Using Hyperlinks

One of the most useful features of ExtenXLS is the ability to dynamically set hyperlinks on Cells in your output files.

This functionality is especially useful in web applications that provide a 'drill-down' view of spreadsheet data. For example, your Servlet may produce a master report with a number of referenced detail reports. You can program your application to create hyperlinks on the detail Cells in your master report, which when clicked, open a request to the server for a detailed information report containing data for the detail record.

The setting of a dynamic URL is a simple method call on the CellHandle:

Example: `myCell.setURL("http://yoursite.com/");`

The following code demonstrates setting the URL on a number of new Cells:

```
try{
    String ht="E3:E10";
    for(int t = 3; t<=10;t++){
        try{
            sheet1.add("ExtenXLS Home Page","E" + t);
            catch(CellPositionConflictException e){System.out.println(e);}
            CellHandle link1 = sheet1.getCell("E"+t);
            sheet1.moveCell(link1,st4 + t);

            link1.setURL("http://www.extentech.com/estore/product_detail.jsp?product_group_id=1");}
        }catch(CellNotFoundException e){System.out.println(e);}
```



Note: Due to the undocumented nature of the underlying HLink spreadsheet record type, not all Hyperlinks contained in existing WorkBooks are supported. A warning will be issued to System.err when these are encountered during parsing of the Workbook.

Template Guidelines

ExtenXLS is often used as a template-based system. Since Excel is a complete report design tool, it often makes sense to define complex formatting, formulas, and Cell ranges using Excel. You can then read this template file using ExtenXLS and modify it programmatically at runtime using the API methods. After your code has modified the Cell values, you can stream the XLS bytes to a file, web browser, or database.

Since the XLS file is a template, you may want to maintain placeholder values in the Cells that you wish to change. Please note, placeholder values are optional, but may aid in visual layout of you reports.

If you do use placeholder values, they should be the appropriate type for the Cell to ensure that ExtenXLS will not waste time converting the data types in the Cell.

Cells do not exist in a spreadsheet file unless there is a value in them. For this reason, if there is no data in the Cell, it is not saved to disk by spreadsheet (for file optimization reasons) and likewise ExtenXLS will not be able to find it. The API will throw a `CellNotFoundException` if you try to access a Cell that using ExtenXLS that does not exist on the template `WorkSheet`.

If you want to work with a Cell in a `WorkBook` that may or may not exist at runtime, simply catch the `CellNotFoundException` and in the catch block call a **`WorkSheetHandle.add(Object r, String address)`** with the new value as your object parameter. You can then safely get a `CellHandle` to the Cell.

Sample Applications

Included in your installation is a sample template XLS file containing a number of example program files and template spreadsheets.

We recommend reading and studying any sample code which might pertain to the application you are developing in order to gain an understanding of the relevant functions and syntax.

Additional Resources

For general information on Java programming the Java Tutorial is a great place to start, quickly followed by Bruce Eckel's invaluable "Thinking in Java" online book.

OpenOffice.org, the Gnome Project and Lotus provide BIFF8 compatible desktop applications.

Troubleshooting

If you are having problems with your reports, please review the table:

Symptom	Possible Solutions
Report output is incorrect; reports are not created, data missing.	Review any System.out and System.err logs and console messages for obvious warnings and errors.
Output reports display many duplicate and random incorrect values scattered about	<p>This is the result of setting the value of a Cell containing a shared String. Please be sure that your template file contains unique placeholder values in any mapped Cells and rows to ensure that you do not set a value (such as an empty space) which is shared between many other Cells.</p> <p>If you are adding Strings to the file, set the <code>WorkbookHandle.setDupeStringMode(WorkbookHandle.ALLOWDUPES);</code></p>

	<p>Please review the section on Important Issues With Strings on page 27</p>
<p>Report output does not match expectations.</p>	<p>When other techniques fail to fix your problems, try the following:</p> <p>Test your report using a completely blank template file. Check the rows/cols and reconcile with expectations.</p> <p>Execute your queries using a SQL administration tool to verify that your data results are as expected.</p>
<p>WorkBookHandle constructors throw License exceptions</p>	<p>ExtenXLS must be able to locate both the location of its class files as well as the extenxls.lic file in order to instantiate a WorkBookHandle.</p> <p>If your error message is:</p> <p>ERROR: Empty or Missing License File.</p> <p>Check to be sure you have a valid license key in your extenxls.lic file and that there is no other text or whitespace in this file</p> <p>Check your classpath for the ExtenXLS.jar file and ensure</p>

	<p>that the extenxls.lic file is in the same directory.</p> <p>If your error message is:</p> <p>ERROR: Invalid License File Detected loading ExtenXLS. Please contact support@extentech.com.</p> <p>Check your classpath for the ExtenXLS.jar file and ensure that the extenxls.lic file is in the same directory.</p> <p>Check that there are no older/other versions of ExtenXLS.jar or ExtenBIS.jar in your classpath.</p> <p>Download the latest version of the software from www.extentech.com and replace the jar with the newest available version.</p> <p>If you have deployed your application using a WAR or EAR file, be sure to set the jarloc and licensekey System properties per the instructions on page 9 of this document.</p>
--	---

More information can be found online by accessing the Extentech knowledgebase at:
<http://www.extentech.com/knowledgebase/KBList.jsp>

If you are a current support customer and have any questions regarding ExtenXLS or need assistance, please call the Extentech support line 9-5pm Pacific Standard Time at:

415-759-5292 or leave a message on our toll free incident report line:

1-800-787-6849. Email support can be found at: support@extentech.com.

If you have any comments or suggestions for improvement of this document, we would like to hear from you. If there is anything you would like to see documented that is not noted here, please email us at: support@extentech.com with your suggestions. Thank you for using ExtenXLS.

A Note about ExtenBIS and the Extentech Roadmap

The Extentech Business Intelligence Server (ExtenBIS™) is built upon the foundation of ExtenXLS, the Luminet™ Java Application Server, as well as the ExtenBEAN™ Object-Relational Framework, and the Data Transformation Engine.

Combining the best features of spreadsheet-based reporting, with the power and cross-platform capability of Java, ExtenBIS™ overcomes the limitations of static file-based reports, by employing a dynamic, secure, Web-based reporting environment based on your legacy spreadsheets.

With ExtenBIS you can create original and flexible feature-rich reports making them instantly available to thousands of users through the Web simply by mapping data to your spreadsheets and deploying them to the ExtenBIS Business Intelligence Server. By combining the power and familiarity of spreadsheets with a powerful, scalable Java application server, ExtenBIS™ can be the perfect tool to unlock the business intelligence potential your organization already has.

ExtenBIS™ provides data transformation and object-relational mapping. ExtenBIS™ also allows you to create dynamic, data-driven sites using Java Servlets, and JSP along with a data integration layer built upon a flexible Object-Relational framework. You can then output your data in dynamic spreadsheet files in custom web applications.

ExtenBIS™ includes a high-performance reporting engine for enhanced scalability, a complete formula engine capable of calculating formulas on the server, as well as redistributable Swing WorkBook components, and source code.

Copyright ©2005, Extentech Inc.

All Rights Reserved. No part of the contents of this document may be reproduced or transmitted in any form or by any means without the written permission of the publisher.

*Excel is a trademark of Microsoft Corporation

**Java and the Java logo are trademarks of Sun Microsystems Inc.

All other copyrights and trademarks are the property of their respective owners.