

Spreadsheet Selector Object

User Guide



Overview	3
Selector Object	4
Properties	4
Methods	4
Files Collection	6
Properties	6
Methods	6
File Object	7
Properties	7
Methods	8
ErrorLog Collection	8
Properties	8
Methods	8
References Collection	9
Properties	9
Methods	9
Reference Object	10
Properties	10
Methods	10
Appendix A : Criteria File Format	11
Appendix B : Example Code	13
Appendix C : Using the Selector for Web-based applications	14

Overview

The Addix Spreadsheet *Selector* Object is able to extract values and formulae directly from Microsoft Excel spreadsheet files without requiring either the use or presence of Excel. In keeping with the highly desirable aim of making applications development both rapid and cost effective, the interface to the application is in the form of a COM object, that can be accessed primarily from Visual Basic and Visual Basic for Applications, but if required, is also capable of being used from Java in web-based applications.

The *Selector* object can extract values or formulae from multiple spreadsheets in a single pass. Information to be selected is described using either standard Excel references or named ranges. The extraction process can be performed either synchronously or asynchronously. Using asynchronous extraction is particularly useful when extracting large volumes of data, because control is returned to the user as soon as the extraction has commenced. This allows work to continue whilst the extraction proceeds as a background process, terminating automatically upon completion. Once the extraction is complete, the extracted data can be returned as either a *Variant* array or an ADO *Recordset*.

Each *Selector* object is organized hierarchically, being composed of one or more *File* objects, with each in turn referring to a specific Excel spreadsheet file. Each *File* object is in turn composed of one or more *Reference* objects. At the lowest level of organization, *Reference* objects describe a specific area within the selected spreadsheet, from which either data values or formulae may be returned.

The properties (and therefore the data extraction criteria) of each *File* object or even the entire *Selector* object, can be read from or written to an external text file using editors as diverse as Microsoft Word or Notepad. This functionality provides a highly flexible and powerful configuration environment, so that both simple and complex data extractions can be developed and maintained quickly and cost-effectively, using syntax familiar to the vast majority of spreadsheet users.

The *Selector* is available directly from the "Products" page on the Addix web site, by simply double-clicking on the file-download icon. The process is fully automatic, with the download process guiding the user through a small number of simple steps. Download times vary depending on the configuration of the machine being used. Installation and inclusion into Visual Basic or Visual Basic for Applications projects is achieved in the familiar fashion.

All requests for information, product suggestions or requests for support, should be directed to:

info@addix.com.

Selector Object

Properties

Name	Type	Access	Description
<i>Files</i>	Collection	-	Collection of <i>File</i> objects
<i>Executing</i>	Boolean	Read only	A flag indicating that the <i>Selector</i> object is executing asynchronously. During an asynchronous operation, any attempt to invoke a method or modify a property will result in a run time error.
<i>Mode</i>	ModeType	Read/Write	Integer constant used to determine operation mode. Possible values are <i>Synchronous</i> or <i>Asynchronous</i> .
<i>LogFile</i>	String	Read/Write	Filename of optional activity log. If only a filename is supplied, or only a relative path is included, a default location of C:\TEMP is assumed. If this property is set to a null string, no activity log is written. Since writing a log file extends the time taken to complete an extraction, this property should be set only when it is necessary to examine the progress of an extraction, or when developing a criteria file.

Methods

Update ()

Parameters

None

Description

Initiates an extraction operation. Invoking the *Update* method will result in any current results being discarded. If the *Mode* property is set to *Synchronous*, this method will not return until the extraction operation has completed. If the *Mode* property is set to *Asynchronous*, the method will return immediately, leaving the extraction operation to run as a background process. This background process can be interrupted using the *Interrupt* method at any point during its operation. Asynchronous operation is particularly useful when extracting large volumes of data, as control is returned to either the user or the calling process immediately upon initialisation of the extraction, thereby minimizing time lost through waiting for the completion of an extraction.

Interrupt ()

Parameters

None

Description

Halts the current synchronous extraction operation being performed by the *Selector* object. This method is particularly useful if premature termination of an extraction is required.

Read (CriteriaFile)

Parameters

CriteriaFile Required. String. Fully qualified path and filename of a criteria file from which to read in data selection criteria.

Description

Reads selection criteria from an external definition file and uses them to create and populate *File* and *Reference* objects within the *Selector* object. Any existing property values will be overwritten by this method. For further details of the format of criteria files, see Appendix 1.

Write (CriteriaFile)**Parameters**

CriteriaFile Required. String. Fully qualified path and filename of a criteria file to which to write out all the selection criteria associated with the current *Selector* object.

Description

Writes the current selection criteria for all data selections contained in the *Selector* object to an external criteria file. For further details of the format of criteria files, please see Appendix 1.

Files Collection

Properties

Name	Type	Access	Description
<i>Count</i>	Long	Read only	Number of <i>File</i> objects in the collection.

Methods

Item (Index)

Parameters

Index Variant. Index of the selected item within the collection or a fully qualified spreadsheet filename.

Description

Locates a *File* object in the collection by ordinal number, or by matching spreadsheet filename and returns its reference.

Add (Filename)

Parameters

Filename String. Fully qualified spreadsheet filename to which the *File* object refers.

Description

Adds a new *File* object to the *Files* collection. This new *File* object contains details of any data to be extracted from the specified spreadsheet file. Attempting to add a *File* object that references a spreadsheet already referenced by an existing *File* object in the collection will result in failure.

Remove (Index)

Parameters

Index Variant. Index of the selected item within the collection.

Description

Locates a *File* object in the collection by ordinal number or by matching the spreadsheet filename. The located object is removed from the collection and subsequently destroyed.

File Object

Properties

Name	Type	Access	Description
<i>Filename</i>	String	Read/Write	Filename and path of spreadsheet file to criteria data from.
<i>References</i>	Collection	-	Collection of <i>Reference</i> objects.
<i>ErrorLog</i>	Collection	-	Collection of <i>Strings</i> detailing any errors encountered during extraction of the data from the target spreadsheet.

Methods

Write (CriteriaFile)

Parameters

CriteriaFile	String.	Fully qualified filename of a criteria file.
--------------	---------	--

Description

Writes the current selection criteria for the spreadsheet referred to by the *File* object to an external criteria file. For further details of the format of criteria files, please see Appendix 1.

ErrorLog Collection

Properties

Name	Type	Access	Description
<i>Count</i>	Long	Read only	Number of error strings in the collection.

Methods

Item (Index)

Parameters

Index Variant. Index of the selected error string within the collection.

Description

Locates an error string in the collection by ordinal number and returns its reference. This is particularly useful as an aid to debugging when developing criteria file as it allows rapid location of user problems.

References Collection

Properties

Name	Type	Access	Description
<i>Count</i>	Long	Read only	Number of <i>Reference</i> objects in the collection.

Methods

Item (Index)

Parameters

Index Long. Index of the selected item within the collection.

Description

Locates a *Reference* object in the collection by ordinal number and returns its reference.

Add (Reference, Rows, Columns)

Parameters

Reference String. A (case insensitive) worksheet-name, named range, or cell reference. For examples, refer to the documentation on the format of criteria files.

Rows String. A selection of row numbers or ranges to apply to the area defined by *Reference*. The numbers should be in the range 1 to 65536 (the maximum number of rows in a sheet) and can appear in any order and any number of times. Ranges can be descending as well as ascending. A question mark can also be used in a range when the first or last row number isn't known. For examples, refer to the documentation on the format of criteria files.

Columns String. One or more column-indices or ranges. This string is applied to the area defined by *Reference*. The indices should be in the range "A" to "IV" and can appear in any order and can be repeated any number of times. Ranges can be descending as well as ascending. A question mark can also be used in a range when the first or last column index is unknown. For examples, refer to the documentation on the format of criteria files.

Description

Adds a new set of data selection criteria to the spreadsheet referenced by the parent *File* object.

Remove (Index)

Parameters

Index Long. Index of the selected item within the collection.

Description

Locates an *Extract* object in the collection by ordinal. The located object is removed from the collection and subsequently destroyed.

Reference Object

Properties

Name	Type	Access	Description
<i>Reference</i>	String	Read/Write	Worksheet name, named range or cell reference from which to select data for subsequent extraction.
<i>Rows</i>	String	Read/Write	Specification of the rows (combining indices and/or ranges) of the <i>Reference</i> from which to select data.
<i>Columns</i>	String	Read/Write	Specification of the columns (combining indices and/or ranges) of the <i>Reference</i> from which to select data.

Methods

Value (Headings)

Parameters

Headings Optional Boolean. Flag indicating inclusion of source row/column indices.

Description

This method returns the data extracted by the *Reference* object in the form of a *Variant* array. Setting the *Headings* parameter to *True* will result in the addition of an extra row and column at the top and left edges of the returned array. The extra row and column will contain the row and column indices from which the data was sourced.

ValueRS ()

Parameters

None

Description

This method returns the data extracted by the *Reference* object in the form of an ADO *Recordset*.

Formula (Headings)

Parameters

ExtractGroup Optional Text. Text value used to identify groups of extracts.

Description

This method returns the formulae extracted by the *Reference* object in the form of a *Variant* array. Setting the *Headings* parameter to *True* will result in the addition of an extra row and column at the top and left edges of the returned array. The extra row and column will contain the row and column indices from which the data was sourced.

FormulaRS ()

Parameters

None

Description

This method returns the formulae extracted by the *Reference* object in the form of an ADO *Recordset*.

Appendix A : Criteria File Format

Valid criteria files consist of valid spreadsheet filenames, valid references and valid comments.

Any line within the criteria file that has a hash as its first character or is composed entirely of white-space is treated as a comment line and ignored. For example:

```
# File generated on 06-Jan-2000 12:43:07
```

Spreadsheet filenames are written within square brackets and should include the drive as well as the full path:

```
[C:\MyFiles\Examine1.xls]
```

This must be followed by any references that apply to the spreadsheet. (References are always applied to the spreadsheet definition immediately preceding the reference).

Each reference line should contain a minimum of one term, but may contain up to three. Each term should be separated by one or more tabs or spaces (but ***not*** commas) and may be enclosed in single or double quotes.

The first term is a case insensitive worksheet name, a named range or a cell reference:

```
RegionalData
```

```
Sheet1
```

```
Sheet1!A22:C37
```

Range and sheet names containing spaces should be enclosed double quotation marks.

```
"Profit and Loss"
```

The second term, if supplied, is a selection of row numbers or ranges to apply to the area defined by the first term. The numbers should be in the range 1 to 65536 and can appear in any order and any number of times.

```
1,3,5-7,9
```

Ranges can be descending as well as ascending. A question mark can also be used in a range when the first or last row number isn't known.

```
3,7-5,10-14
```

```
?-20,22,25-27
```

The third term, if supplied, is a selection of column indices or ranges to apply to the area defined by the first term. The indices should be in the range "A" to "IV" and can appear in any order and any number of times.

```
A,C,E-G,I
```

Ranges can be descending as well as ascending. A question mark can also be used in a range when the first or last column index isn't known.

```
C,G-E,J-N
```

```
?-T,V,Y-AA
```

The row and column selection parameters are applied as relative offsets to the area defined by the first parameter. If the first parameter is a worksheet name, then all three parameters will be used to construct a cell reference. For example

```
Sheet1 3-5 C-E
```

would produce the same result as entering

```
Sheet1!C3:E5
```

However, if the first parameter already includes row and column selections, the additional row and column selections will act relative to the top left of the initially selected area. So

```
Sheet1!:C3:E5 2 B
```

would produce the same result as entering

```
Sheet1!D4
```

since the second row after row 3 is row 4 and the second column after column B is column D. The cardinal rule, therefore, is to remember that the first expression defines an absolute area on the target spreadsheet and the subsequent row and column specifiers are relative selection from that area.

Figure 1. Example of a typical *Selector* criteria file

```
# File generated on 06-Jan-2000 12:43:07
[C:\MyFiles\Examine1.xls]
Sheet1 1,3,5-7,9 C,G-E,J-N
Sheet1!C3:E5

[C:\MyFiles\Examine2.xls]
Sheet1!:C3:E5 2 B
'Foreign Exchange' 5-25 A,B,C
Sheet3 ?-20,22,25-27 A,C,E-G,I
```

Appendix B : Example Code

The following is a simple example of how to call the *Selector*:

```
Sub Current()  
  
    Dim server As Selector  
    Dim element As Variant  
    Dim definition As Long  
    Dim server As Selector  
    Dim download As Recordset  
    Dim records As Long  
    Dim fields As Long  
  
    Set server = New Selector  
  
    server.Files.Add "C:\WORK\Example1.xls"  
    server.Files("C:\WORK\Example1.xls").References.Add "Sheet1", "4-10", "B-C"  
    server.Files("C:\WORK\Example1.xls").References.Add "Sheet2", "20-23", "B-C"  
  
    server.Files.Add "C:\WORK\Example2.xls"  
    server.Files("C:\WORK\Example2.xls").References.Add "Sheet2", "2-15", "B-G"  
  
    server.Files.Add "C:\WORK\Example3.xls"  
    server.Files("C:\WORK\Example3.xls").References.Add "Sheet1", "20-23", "B-C"  
  
    server.Update  
  
    For file = 1 To server.Files.Count  
        For Each element In server.Files(file).ErrorLog  
            MsgBox element  
        Next  
        For definition = 1 To server.Files(file).References.Count  
            For Each element In server.Files(file).References(definition).ErrorLog  
                MsgBox element  
            Next  
  
            Set download = server.Files(file).References(definition).Recordset  
            download.MoveFirst  
            download.MoveLast  
            records = download.RecordCount  
            fields = download.fields.Count  
  
            Set download = Nothing  
        Next  
    Next  
  
    server.Write "C:\Work\Selector.txt"  
  
    server.Files.Remove "C:\WORK\Example1.xls"  
    server.Files.Remove "C:\WORK\Example2.xls"  
    server.Files.Remove "C:\WORK\Example3.xls"  
  
    Set server = Nothing  
  
End Sub
```

Appendix C : Using the Selector for Web-based applications

Using the Selector COM interface from Java

The interface to the *Selector* is a COM object, which means that it is possible to utilize the power of the *Selector* in web-based applications. The Java SDK is the Microsoft produced extension of the Java programming language and provides a suite of extensions to Java that enables programmers to access Microsoft's Java Virtual Machine COM services.

Versions 2 and later of the JSDK provide JActiveX, which when passed the *Selector* type library, will create Java classes for the coclass of the Automation object, as well as Java interfaces for the interfaces to the *Selector*. The classes and interfaces are derived from the JSDK classes used to access the object and in addition have special comments that specify the GUID's of the coclass and interfaces.

For further information on how to write the required code, email Addix Software Consultancy at:

info@addix.com

Using the Selector COM interface with C#

Microsoft's recently announced a new programming language called C# (pronounced C-sharp as in musical notation) which is expected to ship in late 2000 or early 2001. The aim of C# is to provide a new programming paradigm which will combine the power of C/C++ with the ease of use of Visual Basic.

No application is completely future-proof, but as the code for the Selector is written in C++ and accessed using COM, porting applications to the new C# environment should not prove to be problematic. Check the Addix website regularly for news of how the release of C# will affect your applications developed using the Selector.