Operis Analysis Kit 4.4

Version 4.40.035

© 1997-2019 Operis Group plc
OPERIS ANALYSIS KIT

This manual describes Operis Analysis Kit 4.4, which was issued on 24 March 2019.

FEEDBACK

Feedback, including bug reports and comments on the functionality and user interface of OAK, as well as comments regarding the on-line help and this manual can be emailed to oak@operis.com.

© 1997-2019 Operis Group plc

All rights reserved. No parts of this work may be reproduced in any form or by any means - graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems - without the written permission of the publisher.

Products that are referred to in this document may be either trademarks and/or registered trademarks of the respective owners. The publisher and the author make no claim to these trademarks.

While every effort has been made to ensure the accuracy of this publication, it may contain technical or typographical errors. In addition, the information in this document is subject to change without notice. The publisher and the author assume no responsibility for errors or omissions, or for damages resulting from the use of information contained in this document or from the use of programs and source code that may accompany it. In no event shall the publisher and the author be liable for any loss of profit or any other commercial damage caused or alleged to have been caused directly or indirectly by this document.

Printed: March 2019.

PUBLISHER

Operis Business Engineering Limited

MANAGING EDITOR

David Colver
Operis is a leading adviser in project finance.

Working across a broad range of geographic locations and sectors covering natural resources, power and infrastructure, its services include debt and equity raising, financial modelling and tax & accounting advice.

The firm is best known for its practice in the auditing of financial models, on which it is called by the best-known banks as part of their due diligence examination of transactions to which they are contemplating lending.

Established in 1990, Operis is also a leading provider of financial modelling training, imparting its skills to bankers and finance professionals around the world. It is the developer of the Operis Analysis Kit (OAK), a highly regarded software tool for financial model developers and auditors.

Operis Business Engineering Limited is authorised and regulated by the UK’s Financial Conduct Authority.
## Contents

### A

**Getting started with OAK**

- Welcome to OAK ........................................................................................................... 17
  - 1.1 Welcome to OAK ................................................................................................. 18
  - 1.2 What’s new in OAK 4.4 ....................................................................................... 18
  - 1.3 General caution ...................................................................................................... 20

**OAK Help** .................................................................................................................... 21
  - 2.1 Three ways to get help ......................................................................................... 22
  - 2.2 Updating OAK help ............................................................................................. 22
  - 2.3 Example spreadsheets .......................................................................................... 23
  - 2.4 International spelling ............................................................................................ 23
  - 2.5 Repetition .............................................................................................................. 24
  - 2.6 Pagination ............................................................................................................. 25

**Applications for OAK** ................................................................................................ 27
  - 3.1 Applications for OAK .......................................................................................... 28
  - 3.2 Simple spreadsheets ............................................................................................ 28
  - 3.3 Developing spreadsheet models ............................................................................ 28
  - 3.4 Understanding other people’s models .................................................................. 28
  - 3.5 Formally auditing spreadsheets ............................................................................ 29
  - 3.6 Parallel reconstruction .......................................................................................... 29
  - 3.7 Code inspection ..................................................................................................... 30
  - 3.8 Reconciling to a template model .......................................................................... 30

**Installing OAK** ............................................................................................................. 33
  - 4.1 Installing OAK ...................................................................................................... 34
  - 4.2 Getting the OAK software .................................................................................... 34
  - 4.3 Simple installation ................................................................................................. 35
  - 4.4 The Add-Ins Manager .......................................................................................... 35
  - 4.5 Buying and Activating OAK ................................................................................ 39
  - 4.6 Serial Number Exchange .................................................................................... 45
  - 4.7 Transfer a serial number to a different PC .......................................................... 47
  - 4.8 Checking for updates ........................................................................................... 50
  - 4.9 In case of problems .............................................................................................. 52

### B

**Using OAK**

**OAK menus** .................................................................................................................. 55
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>OAK Menus</td>
<td>56</td>
</tr>
<tr>
<td>1.2</td>
<td>Main menus</td>
<td>56</td>
</tr>
<tr>
<td>1.3</td>
<td>Context menu</td>
<td>56</td>
</tr>
<tr>
<td>1.4</td>
<td>Accelerator key and Keytips</td>
<td>57</td>
</tr>
</tbody>
</table>

**OAK commands**

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>OAK commands</td>
<td>59</td>
</tr>
<tr>
<td>2.2</td>
<td>Map</td>
<td>60</td>
</tr>
<tr>
<td>2.3</td>
<td>Compare</td>
<td>Workbooks</td>
</tr>
<tr>
<td>2.4</td>
<td>Compare</td>
<td>Worksheets</td>
</tr>
<tr>
<td>2.5</td>
<td>Compare</td>
<td>Ranges</td>
</tr>
<tr>
<td>2.6</td>
<td>Compare</td>
<td>Remove Compare Modifications</td>
</tr>
<tr>
<td>2.7</td>
<td>Names</td>
<td>Redefine names</td>
</tr>
<tr>
<td>2.8</td>
<td>Names</td>
<td>Recreate names</td>
</tr>
<tr>
<td>2.9</td>
<td>Names</td>
<td>Apply names</td>
</tr>
<tr>
<td>2.10</td>
<td>Names</td>
<td>Deapply names</td>
</tr>
<tr>
<td>2.11</td>
<td>Names</td>
<td>Remove #REF! names</td>
</tr>
<tr>
<td>2.12</td>
<td>Names</td>
<td>Delete names in cells</td>
</tr>
<tr>
<td>2.13</td>
<td>Names</td>
<td>Localize names in cells</td>
</tr>
<tr>
<td>2.14</td>
<td>Names</td>
<td>Build name database</td>
</tr>
<tr>
<td>2.15</td>
<td>Formula</td>
<td>Reconstruct</td>
</tr>
<tr>
<td>2.16</td>
<td>Formula</td>
<td>Optimize</td>
</tr>
<tr>
<td>2.17</td>
<td>Formula</td>
<td>Prune inactive path</td>
</tr>
<tr>
<td>2.18</td>
<td>Formula</td>
<td>Analyze Discrepancies</td>
</tr>
<tr>
<td>2.19</td>
<td>Search Overview</td>
<td>112</td>
</tr>
<tr>
<td>2.20</td>
<td>Search</td>
<td>Arrays</td>
</tr>
<tr>
<td>2.21</td>
<td>Search</td>
<td>Merged cells</td>
</tr>
<tr>
<td>2.22</td>
<td>Search</td>
<td>Primary error cells</td>
</tr>
<tr>
<td>2.23</td>
<td>Search</td>
<td>Hardcoded constants</td>
</tr>
<tr>
<td>2.24</td>
<td>Search</td>
<td>Constant formula cells</td>
</tr>
<tr>
<td>2.25</td>
<td>Search</td>
<td>Unreferenced cells</td>
</tr>
<tr>
<td>2.26</td>
<td>Search</td>
<td>Conditional search</td>
</tr>
<tr>
<td>2.27</td>
<td>Search</td>
<td>References to blank cell</td>
</tr>
<tr>
<td>2.28</td>
<td>Summarize</td>
<td>125</td>
</tr>
<tr>
<td>2.29</td>
<td>Columns and Rows</td>
<td>Insert</td>
</tr>
<tr>
<td>2.30</td>
<td>Columns and Rows</td>
<td>Delete</td>
</tr>
<tr>
<td>2.31</td>
<td>Transpose</td>
<td>134</td>
</tr>
<tr>
<td>2.32</td>
<td>Copy Address</td>
<td>137</td>
</tr>
<tr>
<td>2.33</td>
<td>Copy Literal</td>
<td>138</td>
</tr>
<tr>
<td>2.34</td>
<td>Remove Color Formatting</td>
<td>141</td>
</tr>
</tbody>
</table>
# Contents

## 2.35 Unhide rows and columns ................................................................. 142
## 2.36 Operis Financial Modelling Book ...................................................... 143
## 2.37 Worksheet Manager ........................................................................ 143
  - Managing Worksheets .......................................................................... 144
  - Type Name for Sheets ........................................................................ 148
  - Worksheet Manager Options ............................................................... 151
## 2.38 Options ............................................................................................ 152

### Scripting OAK

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1 Introduction to scripting</td>
<td>158</td>
</tr>
<tr>
<td>3.2 Getting started in scripting</td>
<td>158</td>
</tr>
<tr>
<td>3.3 Making scripting faster</td>
<td>160</td>
</tr>
<tr>
<td>3.4 OAK methods in the object browser</td>
<td>162</td>
</tr>
<tr>
<td>3.5 Script example: Finding redundant spreadsheets</td>
<td>163</td>
</tr>
<tr>
<td>3.6 Colors in compare and map commands</td>
<td>164</td>
</tr>
</tbody>
</table>

## OAK4 Desktop Edition ............................................................................. 167

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1 Using the Desktop Edition</td>
<td>168</td>
</tr>
<tr>
<td>4.2 History and Purpose</td>
<td>170</td>
</tr>
</tbody>
</table>

## Reference material

### Basic concepts in Excel

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1 Basic concepts in Excel</td>
<td>174</td>
</tr>
<tr>
<td>1.2 Multi-area ranges</td>
<td>174</td>
</tr>
<tr>
<td>1.3 Seven ways to identify a cell's precedents</td>
<td>178</td>
</tr>
<tr>
<td>1.4 Used range</td>
<td>180</td>
</tr>
<tr>
<td>1.5 One-Cell Rule</td>
<td>181</td>
</tr>
<tr>
<td>1.6 Merged cells</td>
<td>182</td>
</tr>
<tr>
<td>1.7 Arrays</td>
<td>183</td>
</tr>
</tbody>
</table>

### Excel Names

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1 Overview of Excel names</td>
<td>186</td>
</tr>
<tr>
<td>2.2 If you use Excel names</td>
<td>186</td>
</tr>
<tr>
<td>2.3 If you avoid Excel names</td>
<td>187</td>
</tr>
<tr>
<td>2.4 Names tutorial A: Getting started</td>
<td>187</td>
</tr>
<tr>
<td>2.5 Names tutorial B: Defining names</td>
<td>188</td>
</tr>
<tr>
<td>2.6 Names tutorial C: Using names in formulas</td>
<td>191</td>
</tr>
<tr>
<td>2.7 Names tutorial D: Names adjustment</td>
<td>193</td>
</tr>
<tr>
<td>2.8 Names tutorial E: Apply names</td>
<td>194</td>
</tr>
</tbody>
</table>
### Basic concepts in OAK

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.9</td>
<td>Names tutorial F: Names that identify several cells</td>
<td>195</td>
</tr>
<tr>
<td>2.10</td>
<td>Names tutorial G: Create names</td>
<td>198</td>
</tr>
<tr>
<td>2.11</td>
<td>Names tutorial H: Useful conventions</td>
<td>201</td>
</tr>
<tr>
<td>2.12</td>
<td>Names tutorial I: Row and column matching</td>
<td>203</td>
</tr>
<tr>
<td>2.13</td>
<td>Names tutorial J: Aggregation</td>
<td>205</td>
</tr>
<tr>
<td>2.14</td>
<td>Names tutorial K: Controlling aggregation</td>
<td>206</td>
</tr>
<tr>
<td>2.15</td>
<td>Names tutorial L: Naming constants</td>
<td>207</td>
</tr>
<tr>
<td>2.16</td>
<td>Names tutorial M: Fine print</td>
<td>208</td>
</tr>
<tr>
<td>2.17</td>
<td>Names tutorial N: Summary</td>
<td>211</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Basic concepts in OAK</td>
<td>215</td>
</tr>
<tr>
<td>3.2</td>
<td>Expression Simplification</td>
<td>216</td>
</tr>
<tr>
<td>3.3</td>
<td>What's safe in simplification</td>
<td>216</td>
</tr>
<tr>
<td>3.4</td>
<td>Simplification rules</td>
<td>218</td>
</tr>
<tr>
<td>3.5</td>
<td>Expression Pruning</td>
<td>220</td>
</tr>
<tr>
<td>3.6</td>
<td>Pruning rules</td>
<td>221</td>
</tr>
<tr>
<td>3.7</td>
<td>Difference between simplifying and pruning formulas</td>
<td>225</td>
</tr>
<tr>
<td>3.8</td>
<td>Discrepancy Analysis</td>
<td>226</td>
</tr>
<tr>
<td>3.9</td>
<td>Fan out</td>
<td>228</td>
</tr>
<tr>
<td>3.10</td>
<td>Constraining discrepancy analysis</td>
<td>229</td>
</tr>
<tr>
<td>3.11</td>
<td>Expression pruning in discrepancy analysis</td>
<td>230</td>
</tr>
<tr>
<td>3.12</td>
<td>Large ranges</td>
<td>231</td>
</tr>
<tr>
<td>3.13</td>
<td>Associating references or values</td>
<td>232</td>
</tr>
<tr>
<td>3.14</td>
<td>Suppressing unmatched subtrees</td>
<td>232</td>
</tr>
<tr>
<td>3.15</td>
<td>Formula reconstruction</td>
<td>234</td>
</tr>
<tr>
<td>3.16</td>
<td>Reconstructing multi-sheet formulas</td>
<td>236</td>
</tr>
<tr>
<td>3.17</td>
<td>The context of a reconstruction</td>
<td>237</td>
</tr>
<tr>
<td>3.18</td>
<td>Reconstructing whole calculations</td>
<td>239</td>
</tr>
<tr>
<td>3.19</td>
<td>Timelines in reconstructions</td>
<td>240</td>
</tr>
<tr>
<td>3.20</td>
<td>Frozen rows and columns</td>
<td>241</td>
</tr>
<tr>
<td>3.21</td>
<td>Reconstructing formulas that use names</td>
<td>242</td>
</tr>
<tr>
<td>3.22</td>
<td>Levels of Reconstruction</td>
<td>243</td>
</tr>
<tr>
<td>3.23</td>
<td>Lots of levels</td>
<td>245</td>
</tr>
<tr>
<td>3.24</td>
<td>Mixed levels</td>
<td>246</td>
</tr>
<tr>
<td>3.25</td>
<td>Left-right consistency</td>
<td>247</td>
</tr>
<tr>
<td>3.26</td>
<td>Simplifying reconstructions</td>
<td>252</td>
</tr>
<tr>
<td>3.27</td>
<td>Pruning reconstructions</td>
<td>253</td>
</tr>
<tr>
<td>3.28</td>
<td>Reconstructing SUMs</td>
<td>255</td>
</tr>
<tr>
<td>3.29</td>
<td>Selections suitable for reconstruction</td>
<td>257</td>
</tr>
</tbody>
</table>

© 1997-2019 Operis Group plc
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.30 Long formulas</td>
<td>258</td>
</tr>
<tr>
<td>3.31 Uses for reconstructions</td>
<td>259</td>
</tr>
<tr>
<td>3.32 Difference between reconstruction and discrepancy analysis</td>
<td>260</td>
</tr>
<tr>
<td>3.33 Formula denaming</td>
<td>261</td>
</tr>
<tr>
<td>Spreadsheet quality</td>
<td>263</td>
</tr>
<tr>
<td>4.1 Risk analysis</td>
<td>264</td>
</tr>
<tr>
<td>4.2 Nesting</td>
<td>264</td>
</tr>
<tr>
<td>4.3 OFFSET and INDIRECT</td>
<td>265</td>
</tr>
<tr>
<td>4.4 Obsolete lookups</td>
<td>266</td>
</tr>
<tr>
<td>4.5 IFERROR</td>
<td>266</td>
</tr>
<tr>
<td>4.6 Hardwired constants</td>
<td>266</td>
</tr>
<tr>
<td>4.7 Cell reference notation</td>
<td>267</td>
</tr>
<tr>
<td>4.8 Cell reference profligacy</td>
<td>267</td>
</tr>
<tr>
<td>4.9 Error constants</td>
<td>268</td>
</tr>
<tr>
<td>OAK object reference</td>
<td>271</td>
</tr>
<tr>
<td>5.1 OAK object reference</td>
<td>272</td>
</tr>
<tr>
<td>5.2 AnalyzeDiscrepancies</td>
<td>272</td>
</tr>
<tr>
<td>5.3 ApplyNames</td>
<td>273</td>
</tr>
<tr>
<td>5.4 BuildNameDatabase</td>
<td>274</td>
</tr>
<tr>
<td>5.5 Compare</td>
<td>276</td>
</tr>
<tr>
<td>5.6 CompareAlignment</td>
<td>278</td>
</tr>
<tr>
<td>5.7 CompareContent</td>
<td>279</td>
</tr>
<tr>
<td>5.8 CompareRanges</td>
<td>279</td>
</tr>
<tr>
<td>5.9 CompareRangesWithColors</td>
<td>281</td>
</tr>
<tr>
<td>5.10 CompareWithColors</td>
<td>282</td>
</tr>
<tr>
<td>5.11 CompareWorkbooks</td>
<td>283</td>
</tr>
<tr>
<td>5.12 CompareWorksheets</td>
<td>283</td>
</tr>
<tr>
<td>5.13 CopyAddress</td>
<td>284</td>
</tr>
<tr>
<td>5.14 CopyLiteral</td>
<td>284</td>
</tr>
<tr>
<td>5.15 DeapplyNames</td>
<td>285</td>
</tr>
<tr>
<td>5.16 DeleteNamesInCells</td>
<td>286</td>
</tr>
<tr>
<td>5.17 DeleteRowsOrColumns</td>
<td>287</td>
</tr>
<tr>
<td>5.18 FormatAddress</td>
<td>288</td>
</tr>
<tr>
<td>5.19 FormatAddressDelimiter</td>
<td>289</td>
</tr>
<tr>
<td>5.20 GetVersion</td>
<td>290</td>
</tr>
<tr>
<td>5.21 ICompareResult</td>
<td>290</td>
</tr>
<tr>
<td>5.22 IDeapplyNamesResult</td>
<td>291</td>
</tr>
<tr>
<td>5.23 InsertRowsOrColumns</td>
<td>292</td>
</tr>
<tr>
<td>Section</td>
<td>Function</td>
</tr>
<tr>
<td>---------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>5.24</td>
<td>IOAKAddIn</td>
</tr>
<tr>
<td>5.25</td>
<td>IOAKAPI</td>
</tr>
<tr>
<td>5.26</td>
<td>IRenameFailure</td>
</tr>
<tr>
<td>5.27</td>
<td>IRenameResult</td>
</tr>
<tr>
<td>5.28</td>
<td>LocalizeNamesInCells</td>
</tr>
<tr>
<td>5.29</td>
<td>Map</td>
</tr>
<tr>
<td>5.30</td>
<td>MapComponents</td>
</tr>
<tr>
<td>5.31</td>
<td>MapColor</td>
</tr>
<tr>
<td>5.32</td>
<td>MapOutput</td>
</tr>
<tr>
<td>5.33</td>
<td>OAKResult</td>
</tr>
<tr>
<td>5.34</td>
<td>Operis_OAK</td>
</tr>
<tr>
<td>5.35</td>
<td>Optimize</td>
</tr>
<tr>
<td>5.36</td>
<td>Prune</td>
</tr>
<tr>
<td>5.37</td>
<td>PruningMode</td>
</tr>
<tr>
<td>5.38</td>
<td>Reconstruct</td>
</tr>
<tr>
<td>5.39</td>
<td>Reformulate</td>
</tr>
<tr>
<td>5.40</td>
<td>RecreateNames</td>
</tr>
<tr>
<td>5.41</td>
<td>RedefineName</td>
</tr>
<tr>
<td>5.42</td>
<td>RemoveColorFormatting</td>
</tr>
<tr>
<td>5.43</td>
<td>RemoveHashRefNames</td>
</tr>
<tr>
<td>5.44</td>
<td>RenameErrorType</td>
</tr>
<tr>
<td>5.45</td>
<td>SearchFor</td>
</tr>
<tr>
<td>5.46</td>
<td>SearchForArrays</td>
</tr>
<tr>
<td>5.47</td>
<td>SearchForConstantFormula</td>
</tr>
<tr>
<td>5.48</td>
<td>SearchForHardcodedConstant</td>
</tr>
<tr>
<td>5.49</td>
<td>SearchForMerged</td>
</tr>
<tr>
<td>5.50</td>
<td>SearchForPrimaryError</td>
</tr>
<tr>
<td>5.51</td>
<td>SearchForReferencesToBlankCells</td>
</tr>
<tr>
<td>5.52</td>
<td>SearchForUnreferencedCells</td>
</tr>
<tr>
<td>5.53</td>
<td>SearchForValue</td>
</tr>
<tr>
<td>5.54</td>
<td>ShowProgress</td>
</tr>
<tr>
<td>5.55</td>
<td>Summarize</td>
</tr>
<tr>
<td>5.56</td>
<td>Transpose</td>
</tr>
<tr>
<td>5.57</td>
<td>UndoCompareModifications</td>
</tr>
<tr>
<td>5.58</td>
<td>UnhideCells</td>
</tr>
<tr>
<td>5.59</td>
<td>WorksheetManager</td>
</tr>
</tbody>
</table>

**Known limitations**

<table>
<thead>
<tr>
<th>Section</th>
<th>Section Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.1</td>
<td>Known limitations</td>
<td>333</td>
</tr>
<tr>
<td>6.2</td>
<td>3-D references</td>
<td>334</td>
</tr>
<tr>
<td>6.3</td>
<td>Structured references</td>
<td>334</td>
</tr>
<tr>
<td>6.4</td>
<td>Doubly prefixed references</td>
<td>335</td>
</tr>
<tr>
<td>6.5</td>
<td>Format limits</td>
<td>335</td>
</tr>
<tr>
<td>6.6</td>
<td>Deleting corrupt names</td>
<td>336</td>
</tr>
<tr>
<td>Problem solving</td>
<td>337</td>
<td></td>
</tr>
<tr>
<td>7.1</td>
<td>Problem solving</td>
<td>338</td>
</tr>
<tr>
<td>7.2</td>
<td>Forcing preinstallation testing</td>
<td>338</td>
</tr>
<tr>
<td>7.3</td>
<td>Common problems</td>
<td>338</td>
</tr>
<tr>
<td>7.4</td>
<td>Reporting bugs</td>
<td>339</td>
</tr>
<tr>
<td>7.5</td>
<td>Finding where OAK is installed</td>
<td>340</td>
</tr>
<tr>
<td>7.6</td>
<td>Disabled Items</td>
<td>341</td>
</tr>
<tr>
<td>7.7</td>
<td>Licensing</td>
<td>342</td>
</tr>
<tr>
<td>7.8</td>
<td>COM Registrations</td>
<td>344</td>
</tr>
<tr>
<td>7.9</td>
<td>OAK Registry Settings</td>
<td>345</td>
</tr>
<tr>
<td>Resources</td>
<td>351</td>
<td></td>
</tr>
<tr>
<td>8.1</td>
<td>Useful spreadsheet resources</td>
<td>352</td>
</tr>
<tr>
<td>8.2</td>
<td>Eusprig</td>
<td>352</td>
</tr>
<tr>
<td>8.3</td>
<td>Ray Panko’s spreadsheet research website</td>
<td>352</td>
</tr>
<tr>
<td>8.4</td>
<td>Microsoft Excel blog</td>
<td>352</td>
</tr>
<tr>
<td>Version histories</td>
<td>355</td>
<td></td>
</tr>
<tr>
<td>9.1</td>
<td>OAK version history</td>
<td>356</td>
</tr>
<tr>
<td>9.2</td>
<td>Version history for this help</td>
<td>369</td>
</tr>
<tr>
<td>Index</td>
<td>375</td>
<td></td>
</tr>
</tbody>
</table>
Getting started with OAK
Welcome to OAK
1.1 Welcome to OAK

OAK, the Operis Analysis Kit, adds capabilities to Microsoft Excel that are useful to people who
- develop substantial spreadsheet models
- need quickly to understand existing models
- are formally auditing other people’s models
- need to enforce compliance with regimes that control spreadsheet use, including Sarbanes Oxley.

This help sets out
- an overview of OAK, its philosophy and its capabilities
- what's new in OAK 4.4, and what is special about it
- what you need to run OAK
- details of all the OAK commands: what they do and when you might use them
- tutorials on how to use OAK to audit a spreadsheet, and how to extend its capabilities by programming it.

1.2 What's new in OAK 4.4

The main innovation in OAK 4.4 is the splitting of the product between
- OAK Professional, with the same capabilities as before
- OAK Essentials, which makes the four most popular OAK features at a more affordable price.

<table>
<thead>
<tr>
<th>Capacity</th>
<th>OAK ESSENTIALS</th>
<th>OAK PROFESSIONAL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Workbooks of up to 10 worksheets</td>
<td>Unlimited</td>
</tr>
</tbody>
</table>

Features common to both products

<table>
<thead>
<tr>
<th>Summarize workbook</th>
<th>Summarize workbook</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map workbook</td>
<td>Map workbook</td>
</tr>
<tr>
<td>Search</td>
<td>Search</td>
</tr>
<tr>
<td>Compare</td>
<td>Compare</td>
</tr>
<tr>
<td>Help</td>
<td>Help</td>
</tr>
</tbody>
</table>
Additional features of OAK Professional

Cells
- Transpose
- Copy literal
- Copy address

Worksheet
- Insert / delete rows and columns
- Unhide cells
- Remove color formatting

Names
- Apply
- Redefine
- Localize
- DeApply
- ReCreate
- Delete
- Remove #REF!
- Build database

Formula
- Optimize
- Reconstruct
- Prune inactive path

Worksheet manager

Operis news
Present
Absent

VBA / COM interface
Present, but not functional
Fully functional

Memory-saving OAK Desktop Edition
Not provided
Provided

Help: online, offline and PDF manual
50 pages
350 pages

The two products are the same program. The set of features activated by the software is dictated by the licensing file present on the workstation. Upgrading from OAK Essentials to OAK Professional is just a matter of updating the license: no new software needs loading.

Some significant technical changes have been made internally to the product. These, and a log of changes from release to release of version 4 can be found in the reference material.
In particular:

The Compare workbooks functionality compares A1 or R1C1 formulas.

The Map workbooks command can now color range consistency with different colors for each direction.

The Summarize workbooks command includes a cross reference report to show the number of linkages between worksheets.

On a more technical note:

OAK now uses the .NET Framework 4.5.2

OAK no longer requires Microsoft Office Primary Interop Assemblies installed.

1.3 General caution

Many of OAK’s commands alter the active worksheet, and no Undo option is offered for any of them. (Sadly, Microsoft doesn't expose the mechanisms that would be necessary to allow an arbitrary sequence of Excel and Add-in actions to be reversed systematically. If it did, Operis would implement Undo in OAK.)

OAK intentionally does not deliver nagging messages warning that it is making hard-to-reverse changes to what might be the only copy of your spreadsheet. Operis aims OAK at competent spreadsheet users, and leaves it to them to take appropriate precautions, including

- making spreadsheet documents read-only before starting to work on them, so that it is not possible unintentionally to overwrite them with versions that OAK has changed
- saving work frequently
- including some kind of version number in the file names, and increasing it at intervals, so that there is a chronological audit trail of different versions of the model.
OAK Help
2.1 Three ways to get help

Operis aims to make this help a valuable resource for Excel users, whether or not they are users of OAK. It is accessible in three ways.

- It is shipped with OAK in the .CHM Windows help file format.
- It is shipped with OAK as a manual in PDF format.
- There is also a version online, at www.operisanalysiskit.com.

LOCAL HELP

When OAK is installed on your computer, local version of the help is provided in a file called OAK4.CHM.

Each page in the CHM version of the help has a link at the top "This page online". Assuming the internet is reachable from your location, this link will take you to the www.operisanalysiskit.com version of the page you are looking at.

WEB-BASED HELP

Versions of OAK help are also available online at www.operisanalysiskit.com. You need to be connected to the internet to reach this version, but it may be more up to date than the one you have on your computer.

PRINTED MANUALS

The PDF version of the OAK help is called OAK4.pdf. It is handy if you want to occupy your travel time by reading it from cover to cover on the train.

2.2 Updating OAK help

A comparison of local and web based pages of the OAK help may show that the help on www.operisanalysiskit.com is a more recent version than the one you have installed on your workstation.

In such cases you are free to download the newer files and install them.

1. Download the latest OAK help CHM file.
2. Download the latest OAK help PDF file.
3. Drag the both downloaded files into the Help subdirectory of the directory in which OAK is installed, and allow Windows to replace the older equivalent files.

Tip: The directory in which OAK is installed is usually C:\Program Files\Operis\OAK4, making the folder required here C:\Program Files\Operis\OAK4\Help; but it might be somewhere else if the defaults offered by the setup program were overridden, in which case instructions for finding it can be found in the chapter on Problem Solving in the Reference material.
This version of the help is 4.40.035.

2.3 Example spreadsheets

LOCATION

At various points in this help, readers are invited to load example spreadsheets. These are listed in the following table.

TABLE OF EXAMPLE SPREADSHEETS

<table>
<thead>
<tr>
<th>File</th>
<th>Narrative</th>
</tr>
</thead>
<tbody>
<tr>
<td>Names tutorial B Defining names.xls</td>
<td>Names tutorial B: Defining names</td>
</tr>
<tr>
<td>Names tutorial F Multi cell names.xls</td>
<td>Names tutorial F: Multi cell names</td>
</tr>
<tr>
<td>Names tutorial G Create names.xls</td>
<td>Names tutorial G: Create names</td>
</tr>
<tr>
<td>Names tutorial H Conventions with names.xls</td>
<td>Names tutorial H: Useful Conventions</td>
</tr>
<tr>
<td>Discrepancy analysis example.xls</td>
<td>Discrepancy analysis</td>
</tr>
</tbody>
</table>

2.4 International spelling

INTERNATIONAL AUDIENCES

To make OAK accessible to the widest possible audience, Operis has adopted international conventions for spelling certain words throughout the OAK product and documentation. For this reason you will find

- color rather than colour
- formulas rather than formulae
- optimize rather than optimise.

UK audiences should not be offended by these spellings since OAK is only relevant to users who work closely with Microsoft Excel, and that product has used the same approach throughout its life.

FINANCIAL MODELLING OR MODELING

In British English the phrase financial modelling is spelt with two Ls but the convention in US English is to spell financial modeling with only one L. There is an obstacle to adopting the US spelling in the context of OAK. The two spellings return quite different results in an
internet search, reflecting the fact that the terms are used differently in Europe/Australia and in the US, a divergence in meaning that is growing to the point where reviews on one side of the Atlantic of books written on the other have in effect said "That's not financial modelling/modeling".

Common practice is different too: what is considered unprofessional practice in Europe, and would cost analysts their job, is the method carefully taught in US business schools.

In this documentation the issue has been avoided altogether in favor of some equivalent such as "developing spreadsheet models".

<table>
<thead>
<tr>
<th>Location</th>
<th>Europe and Australia</th>
<th>US</th>
</tr>
</thead>
<tbody>
<tr>
<td>Term</td>
<td>Financial modelling</td>
<td>Financial modeling</td>
</tr>
<tr>
<td>Meaning</td>
<td>The development of spreadsheets to structure and illustrate transactions undergoing financing. Usually involves the projection of traditional financial statements and accompanying ratios.</td>
<td>Covers financial projections also, but increasingly identified with the mathematical analysis of derivative instruments.</td>
</tr>
<tr>
<td>Circularity in models</td>
<td>Always deprecated</td>
<td>Routinely taught in business school classes as a legitimate spreadsheeting technique.</td>
</tr>
<tr>
<td>Balance sheet</td>
<td>Calculate assets and liabilities independently. Include a test that the resulting balance sheet balances as part of the model's internal quality control.</td>
<td>Deriving cash as the balancing item is routinely taught in the best business schools as a legitimate technique.</td>
</tr>
<tr>
<td>Formal model audit</td>
<td>Provided by Operis and by many large accounting firms. A well established step in bringing a large financing to close.</td>
<td>Rarely performed, though beginning to appear.</td>
</tr>
</tbody>
</table>

### 2.5 Repetition

Though Operis has done its best to make this manual easy to read, at some points it may appear to be repeating material that has been covered already. This is because it is assembled by collating the articles that make up the OAK offline and online help.

Many of those articles have a paragraph or two of introduction that serve to set the context for what follows, so that each item is readable as a freestanding text. This scene setting may prove repetitive when related subjects fall on the page very close to each other.
### 2.6 Pagination

It is usual for what is marked as page 1 of a book to be the first page of the text. Title pages, forewords, tables of contents and tearful acknowledgements to one's family fall before page 1.

But in a PDF document, pages are counted from the start. The title page is page 1; the first page of the text lies on page 11. The practical result is that if you ask a PDF reader to take you to what is labelled page 50, you will be taken to a point in the document several pages before that.

To minimize confusion, the pages in this document are numbered from the very start of the document. Page 1 is the title page, and pages 5-10 are the contents listing. By the time we get to what would be page 1 of a traditional publication, the page number is already showing at 11. That's not the traditional way to number the pages, but it does have the advantage that if you ask to go to a particular page, your computer will take you there.
Applications for OAK
3.1 Applications for OAK

Different people use spreadsheets in different ways.

- The overwhelming majority of people only do simple things with spreadsheets: little more elaborate than managing lists, of, for example, the work rota or holiday plans for their team members, or checklists of various kinds.

- Some people will be preoccupied by developing significant spreadsheets.

- Some people will be seeking quickly to understand spreadsheets that other people have sent them.

- Other people will be seeking to do thorough checks of spreadsheets, which may go as far as a full formal audit.

3.2 Simple spreadsheets

The overwhelming majority of people only do simple things with spreadsheets: little more elaborate than managing lists, of, for example, the work rota or holiday plans for their team members, or checklists of various kinds.

These people will find OAK’s ability to compare spreadsheets useful. It is the most powerful way to see what has changed in different versions of the documents they prepare.

3.3 Developing spreadsheet models

Many OAK users will be preoccupied by developing significant spreadsheets. Steps in the process where OAK may be useful include

1. Summarize workbook, to deliver a list of all the values that serve as inputs to the model. These are candidates for inclusion in any documentation that is being prepared to accompany the model.

2. OAK’s various extensions to Excel’s facilities for manipulating names, that address several obstacles that deter many analysts from expressing models in meaningful names rather than less meaningful coordinates.

3. Compare workbooks/worksheets, to establish a version history if one has not been maintained during the development.

3.4 Understanding other people’s models

Many more senior spreadsheet users don’t want to develop models themselves, nor do they have time to check every cell in them, but they do want a means of understanding quickly what is going on inside them and understanding the key calculations.

1. Summarize workbook, which will prepare a table providing the main statistics of the workbook: number of worksheets, distinct formulas and so on. This gives some idea of how big a spreadsheet is
2 OAK can prepare a map of the workbook, to give a high level view of how formulas have been copied across the worksheet.

3 OAK can prepare concise reconstructions of fragments of the spreadsheet that perform particular calculations, giving some idea of what a segment of a model is seeking to do, and identify the inputs it uses.

3.5 Formally auditing spreadsheets

Before it advances money to an organization that comes to it seeking a loan to pursue a particular venture, a bank will often insist that the financial projections that have been presented to it do in fact add up.

The largest part of the work of Operis, the company that developed OAK, is performing financial model reviews of this kind. Many of the larger accounting firms have departments that specialize in auditing spreadsheets too.

The final outcome of getting a spreadsheet checked by any of these organizations is a formal letter expressing an opinion that the spreadsheet does what it is supposed to. But the processes by which they arrive at this opinion vary considerably among the firms.

· Most firms perform a code inspection: they examine the spreadsheet formulas, one at a time.

· At least one firm develops a completely new model from first principles, and reconciles its results to the results provided by its client.

· Operis’s own method is close to that, but much less expensive: it develops parallel reconstructions of the major elements of the model, such as the revenues, costs and interest calculations.

· Yet another firm simply keys the assumptions into its own template model, and sees if it gets similar answers.

OAK provides capabilities that have a role in all of these approaches.

3.6 Parallel reconstruction

Some auditors verify a spreadsheet model by building a new one and reconciling the differences between the two.

Steps in the process where OAK may be useful are

1 Summarize workbook, which will prepare a table providing the main statistics of the workbook: number of worksheets, distinct formulas and so on. This is valuable in estimating how much effort will be involved in reviewing the spreadsheet.

2 OAK can prepare concise reconstructions of fragments of the spreadsheet that perform particular calculations, giving some idea of what a segment of a model is
seeking to do, and identify the inputs it uses, which can guide the reconstruction process.

3 Where the rebuilt model gives a different answer from the one being tested, OAK can analyze the discrepancy between the two and identify its origin.

4 Summarize workbook will deliver a list of all the values that serve as inputs to the model. These are candidates for checking one by one against any model data book or other description that has been provided of the model's assumptions.

5 Compare workbooks/worksheets, to understand how a new version of the model, which needs to be checked, differs from an older version that has been checked already.

3.7 Code inspection

The majority of spreadsheet auditors perform a code inspection: they examine a spreadsheet's formulas, one at a time.

Steps in the process where OAK may be useful are

1 Summarize workbook will prepare a table providing the main statistics of the workbook: number of worksheets, distinct formulas and so on. This is valuable in estimating how much effort will be involved in reviewing the spreadsheet.

2 Summarize workbook will also prepare a list of all the distinct formulas in the workbook, which can be divided up among the team members who are going to perform the inspection.

3 Code inspectors often like to map the workbook, to give a high level view of how formulas have been copied across the worksheet.

4 Build names database will provide a listing of any names that are defined in the workbook. Many people who perform a code inspection feel the need to check the definition of every one of these names. Alternatively, OAK can get rid of the names, leaving a coordinate-only version of the model to be checked.

5 Summarize workbook will deliver a list of all the values that serve as inputs to the model. These are candidates for checking one by one against any model data book or other description that has been provided of the model's assumptions.

6 Compare workbooks/worksheets, to understand how a new version of the model, which needs to be checked, differs from an older version that has been checked already.

3.8 Reconciling to a template model

At least one spreadsheet auditor simply keys the assumptions into its own template model, and sees if it gets similar answers.
Steps in the process where OAK may be useful include

1 Summarize workbook will deliver a list of all the values that serve as inputs to the model. These are candidates for inserting into the template model.

2 Where the rebuilt model gives a different answer from the one being tested, OAK can analyse the discrepancy between the two and identify its origin.

3 Compare workbooks/worksheets, to understand how a new version of the model, which needs to be checked, differs from an older version that has been checked already.
Installing OAK
4.1 Installing OAK

A modern system running Excel 2016 and Windows 10 (modern for a typical OAK user of early-2018), likely already has installed everything that OAK needs. However, the prerequisites are listed here for exceptional cases.

WHAT YOU NEED TO RUN OAK

OAK will run under
- Windows Vista SP2
- Windows 7 SP1
- Windows 8
- Windows 8.1
- Windows 10

The operating system needs to have the following components installed:
- version 4.5.2 of Microsoft's .NET framework

.NET 4.5.2

- is added to Windows 8.1, Windows 8, Windows 7 SP1 and Windows Vista SP2 by you downloading and installing it.

OAK will run under
- Excel 2007
- Excel 2010 32 and 64-bit editions
- Excel 2013 32 and 64-bit editions
- Excel 2016 32 and 64-bit editions

Though there is a version of Excel for the Apple Macintosh, OAK is not designed to work with it.

4.2 Getting the OAK software

Like many applications, OAK is provided as a single .msi file which is used by Windows Installer, Microsoft's software installer that is part of Windows.

To install OAK, you need a copy of OAK4-\version .msi.
The file can be downloaded from www.operisanalysiskit.com/TryOak.htm.

Alternatively, it's perfectly fine to take a copy of that file from a colleague who has already downloaded it.

If your firewall obstructs you from downloading executable files of this kind, Operis will be happy to email it to you on request to oak@operis.com.

Copies on a mini-CD (or a full sized CD if your computer has a slot-loader for discs) are also available from Operis.

PREREQUISITES

OAK has a number of prerequisites, but these are most likely installed already, and the OAK installer will complain if they are not. The prerequisites are listed in the section titled Installing OAK.

4.3 Simple installation

To install OAK
1. Obtain the installer: OAK4-\version\msi
2. Double click on the file, an action that will invoke the Microsoft Software Installer built into Windows.
3. When the installer reports that it has finished, start Excel and see whether the OAK menus or ribbon tabs have appeared.

Most of the time, this is all that is necessary to get a working version of OAK. If it doesn't work first time, the next section will help you fix it.

4.4 The Add-Ins Manager

DISABLING OAK

Operis hopes that OAK will not interfere with any other Excel add in that you may have installed on your computer. If it does, Operis would like to hear about it.

If a conflict does arise, you may wish to disable OAK temporarily when you are using the conflicting add-in.
Getting started with OAK: Installing OAK

The way add-ins are managed has changed in Excel 2007. Earlier versions of Excel had a manager for XLA, XLL and automation add-ins, and a manager for COM add-ins that was separate and harder to find. Excel 2007 and later offer a single add-in manager that lists installed add-ins of all types in one place. While the manager itself is very similar between Excel 2007 and Excel 2013-2016, getting to it is different in each.

Go to Office Button | Excel Options | Add-Ins

For the purposes of accessing the add-ins manager, the user interface is much the same from there, for Excel 2007 to 2016. Just click the Add-Ins item on the left.
In order to enable or disable OAK, locate the drop-down list near the bottom of the Excel Options window, and select "COM Add-ins", then click the "Go" button.

This shows the COM Add-Ins dialog. Again, observe the Operis Analysis Kit 4 add-in.

OAK can be disabled using this dialog, and re-enabled again later when you have finished working with any conflicting product.

**INSTALLING OAK 4 IN THE COM ADD-INS LIST**

If the installation process has worked as intended, OAK will be automatically added to each user's COM Add-ins list in Excel. However, in systems that are severely locked down, usually for security purposes, the installer may be prevented from doing this. In this case, it will be necessary for each user of the machine to add OAK to the COM Add-ins list manually.

1. Go to the COM Add-ins dialog as described in the previous section. Click the Add button, and navigate to the folder where OAK was installed. By default, this is C:\Program Files\Operis\OAK4.

2. Select Operis.OAK.AddIn.dll (or Operis.OAK.AddIn64.dll for the 64-bit version) then click the "OK" button.
Getting started with OAK: Installing OAK

3 Operis Analysis Kit 4 should now appear in the list.
4 Press the "OK" button on the COM Add-ins dialog.
5 After a brief pause, the OAK 4 splash screen should appear briefly.

ENABLING THE DEVELOPER TAB IN EXCEL 2010-2016

If you are using Excel 2010-2016, and you have a need to frequently disable and enable the various COM add-ins (or other add-ins) you use, you may find it helpful to enable the developer tab. This tab contains short cuts to both of Excel's add-in managers. To enable the developer tab, go to the Excel options screen as for the Add-ins manager, but instead of clicking Add-Ins, click Customize Ribbon. Locate "Developer" in the checklist box on the right and click it to enable the developer tab.

Click OK to return to the workspace.
If you need to switch OAK, or any other COM add-in on or off, you can now do this by clicking the Developer tab, then the COM Add-Ins button, rather than the journey through the options.

### 4.5 Buying and Activating OAK

OAK has a 30-day trial period in which it can be evaluated. During this time, it will function fully, reminding the user once per session. When the trial period is up, the only part of OAK that will work is the part that takes licence details.

Licensing is performed on the Licence Information window, which is accessed by going to the Help group in either of the OAK ribbon tabs, and clicking the "Licence Information" item under the Operis menu.

The Licence Information window looks like this:

If you do not wish to enter a licence key, click the cancel button or close box to continue.

Licences can be purchased by contacting Operis using the contact details on the form, or online by clicking the "Purchase" button. This will direct your web browser to a purchase page on the Operis Analysis Kit website, which will direct you to an e-commerce provider.
Getting started with OAK: Installing OAK

where you can purchase a license for OAK. The e-commerce provider will send you a serial number.

When the serial number has been received, enter into the Serial Number text box on the "Single License" tab on the License Information window.

This serial number requires activation. While there is an email-based alternative, this involves OAK contacting a web server on the internet and associating the PC that OAK is installed on with the serial number.

To start the activation process, click the "Activate" button. In the form provided, enter a contact email address, and repeat it to confirm.
Once the email address has been entered, click Done.

This leads to the activation form. OAK offers two means of activation, online, and by email. If you have an internet connection, you are recommended to use online activation as that is automated. If you don't have an internet connection, or OAK is blocked by a corporate firewall or similar, email activation is an alternative (which does not strictly need to be done online).

ONLINE ACTIVATION

Click the "Activate Online" button. OAK will attempt to contact the activation server. This should take a few seconds. If it is successful, you will be shown an information dialog, then returned to the License Information window, which will no longer have the "activation required" message in red.

If OAK fails to connect to the activation server, this may be due to a proxy server being in use. You can make OAK use the proxy server by clicking the "Proxy Settings" button and configuring the options there. You may need to contact your IT support staff to help with filling in the details.
EMAIL ACTIVATION

In some organizations, OAK might be blocked from accessing the activation server, but email still available. For this scenario, there is the option of email activation. To use this, go to the "Activation by Email" tab.

Activation by email involves sending an email to Operis with a hardware identifier for your PC, the same hardware identifier that is sent to the activation server when online activation is used. This email will be processed, and an activation key will be emailed back to you.

The form offers two ways to create an email.

Create: OAK interacts with a desktop email client such as Microsoft Outlook or IBM Notes, to create the email for you to send.
Copy: OAK populates the clipboard with the address, subject and content of the email. You then paste this information into a new email in your webmail provider website. An example of the generated text follows:

To:oak@operis.com
Subject:OAK4 Activation by Email
Body:Product: OPERIS-ANALYSIS-KIT
Serial Number: Pr8kQ-51EE9-6Dc4F-ADi1o-e4T3F-A5F9K
Hardware ID: 096A29E9

Whichever of these options you choose, you must eventually send the email. This email will be processed manually, and an activation key will be generated and emailed to you. This will not be immediate, so you are recommended, at this point, to click the cancel buttons to get back to Excel.

PHONE ACTIVATION

Another option is to call Operis, ask to speak to a member of the OAK sales team, and be ready to quote your serial number and hardware ID from the license screen. You will be provided with an activation key.

EMAIL AND PHONE ACTIVATION: APPLYING AN ACTIVATION KEY

Once you have received the activation key, go to Operis | License Information | Activation tab.
Click the "Apply activation key" button, copy and paste the activation key from your email to the text box in the dialog.

Once you have done this, click OK.

**ACTIVATION COMPLETE**

Once activation is complete, you are then presented with a dialog indicating that.

Click OK to return to the License Information window.
Note that the "Activate" button is now disabled, and the "Transfer" button is enabled. If you ever want to use your serial number on a different computer, you can use the transfer feature to contact the activation server and disassociate your serial number with that PC. You can then reuse your license key on a different PC.

4.6 Serial Number Exchange

Operis sells OAK licenses by the major version number. So if you bought a license for OAK 4.0, that is valid for all OAK 4.x.

OAK 4.2 and earlier used different license keys and purchase procedure. Purchasing and license activation was not automated, and when a user needed to move OAK to a new PC, he or she would have to contact Operis to get a new license key.

OAK 4.3 introduced a licensing system that is integrated with an e-commerce provider, to automate purchasing and serial number (license) activation. Even the process of transferring a license to a different PC is automated, when there is an internet connection that can be used.

For this reason, if you have purchased a license for OAK4 previously, you might want to exchange your legacy license key for a serial number.

As of version 4.4, OAK no longer supports the old (legacy) licenses.

EMAIL

A user with a legacy license can request a serial number from Operis by providing the existing license key. Simply enter the old license key in the serial number box on the license form, click Activate, and follow the prompts.
Getting started with OAK: Installing OAK

The process is to send an email to the OAK support email address with the license key in the subject. This will be processed by Operis's sales department, and a serial number emailed to you.

To create the email in your default email client, such as Microsoft Outlook or IBM Notes, click the "Create" button.

For webmail, click the "To:" and "Subject:" buttons to copy the To: address and subject into your webmail.

TELEPHONE

Alternatively, you could simply call Operis and request to exchange your license key for a serial number. You will need to have your license key ready to read out.

If you know you are not going to be able to activate online, you are recommended to request activation by phone at this time.

ACTIVATION

Once you have received a serial number from Operis, you need to activate the serial number. This is documented in the section titled Buying OAK.
4.7 Transfer a serial number to a different PC

When you buy OAK, you receive a serial number which you then activate to associate your OAK installation with the serial number. This is done online or by email or phone.

Eventually, you might need to install OAK on a different computer. You won't be able to reuse your serial number on that computer unless you deactivate it first.

To do this, go to the License Information window, which is accessed by going to the Help group in either of the OAK ribbon tabs, and clicking the "Licence Information" item under the Operis menu.

On the License Information window, click the "Transfer" button.

Click the "Transfer" button.
Getting started with OAK: Installing OAK

If necessary, proxy settings can be configured by clicking the "Proxy Settings" button.

Click "Deactivate Online". OAK will attempt to contact the activation server and deactivate the serial number.

Click OK to return to the License Information window.
This serial number can now be used on another computer.

PROBLEM - CANNOT CONTACT THE ACTIVATION SERVER

If OAK cannot contact the activation server, you will get a message to indicate this.

After clicking OK, you can click the "Proxy Settings" button to configure proxy settings. You may need to ask your organization's IT staff to help you with this. If you set the proxy settings, and retry the deactivation and you still get this error connecting to the activation server, you are recommended to click "Deactivate by Email" to construct an email to send to OAK support.
4.8 Checking for updates

UPDATE PROGRAM

Operis occasionally releases updates to OAK 4, in the form of additional features and small revisions. OAK can detect when these are available, both automatically when it starts up, or by user request. These updates are free.

OAK refers to new releases within the current major version number as updates, free to licence users of the current OAK installation. New releases where the major version number increases are referred to as upgrades, a new product that requires a new licence purchase.

Updates can consist of a minor version update, e.g. 4.00 to 4.10, which would typically involve the addition of new features, and revision updates, e.g. 4.00.000 to 4.00.003 which would include minor changes.

AUTOMATIC UPDATE CHECKING

By default, periodically when started, OAK will try to access the internet to discover if any updates have been released for OAK. If an update is found, the information is displayed in a form that shows the current version of OAK, and any updates or upgrades that are available.

Note: OAK 5.0 is NOT currently available. This image is for illustration only.

Since this happens when OAK starts up, which is most often when Excel starts up, there is a countdown timer on the OK button, to prevent OAK from blocking Excel's startup procedures waiting for user input. When the timer reaches 0, the form will be closed as...
though the OK button was clicked. The countdown can be stopped by clicking the Stop Countdown button.

If an upgrade to a newer product is found to be available, a checkbox labeled “Do not notify me of this upgrade again” is shown. If this box is checked, only updates will be shown until the next minor version update of the newer product.

CHECKING FOR UPDATES MANUALLY

A user can also check for updates manually. To do this, click OAK Development/Review | Operis | Check for Updates.

The frequency of automatic update checks can be changed by changing the number in the box at the bottom of the form; automatic update checking can be turned off completely by unchecking the check box.

If any updates are available, details of the changes that have been made are shown in the box in the middle of the window. You can download the update and initiate the installation process by highlighting the update in the list and clicking the Update button. Running the installer will require administrator privileges.

Since OAK runs inside Excel, the update check requires Excel to be allowed to make web (http) requests on port 80. Some security configurations may prevent this, and would require reconfiguration of the firewall software.
4.9 In case of problems

BAD DOWNLOAD: "NOT A VALID WIN32 EXECUTABLE"

A common reason for OAK to fail to install is that the file that has been downloaded is incomplete, or, less often, corrupted. The usual signal that this is the case is that your computer complains that the file is "not a valid WIN/32 executable". There is material at www.operisanalysiskit.com/TryOak.htm offering various alternative methods of downloading the file if this proves a problem.

GOOD DOWNLOAD: NO ACTION OR SELF TESTS FAIL

Some workstations, particularly ones that are locked down in secure environments, may present obstacles to the normal installation process. You may be told directly that you don't have the necessary privileges to install software on that machine, and your computer won’t run the OAK installer at all. Alternatively, the installation may appear to happen, but be prevented from doing some necessary thing when it tries to.

This has proved to be a significant issue among the many users of the OAK3, who operate in security conscious businesses in financial services. The setup program of OAK4 has therefore been engineered to perform a variety of preinstallation tests, to inspect the environment in which it finds itself and form an impression (but not a certainty) of the likelihood that OAK will run without problems.

If one or more of those tests fail, the OAK installer will display the results of all of the tests.

Operis will likely ask to see the output from the installer in preinstallation test mode where users are having difficulty installing the product; or an organization's IT department may find it gives all that is needed to overcome installation obstacles. A text summary of the tests suitable for emailing can be generated by pressing the Report button.
Using OAK
1.1 OAK Menus

The main indication that OAK Professional has been successfully installed on a workstation is that the OAK Development and Review tabs appear in the ribbon.

1.2 Main menus

OAK Professional installs two entries on the Excel ribbon.

One features OAK commands that are useful when developing spreadsheet models.

The other features OAK commands that are useful when reviewing spreadsheet models that have already been developed.

Some commands appear on both tabs.

If you prefer, you can switch to the drop down menu on the Add Ins tab, using OAK’s Options command.

1.3 Context menu

OAK Professional presents a concise version of the main menu as a context menu which is presented when the right mouse button is clicked.
1.4 Accelerator key and Keytips

The OAK Professional ribbon tabs can be accessed using the Alt+KD and Alt+KR keytips.

Alternatively, if the OAK Options command has been used to specify that the menu will be used instead, the Add-Ins tab can be accessed using Alt+X, then the OAK menu activated using the A key. If another add-in uses the same keytip, it is possible that Excel will assign OAK a different one. You can change the desired keytip to K in using OAK’s Options command.
OAK commands
2.1 OAK commands

This section of the help documents systematically the commands offered by OAK.

They are presented in the order that they appear in the Add Ins tab menu that used to be used for Excel 2003 and earlier.

Each command is described with

- a sentence or two of introduction, setting out briefly what the command does
- a description of why you might want to use the command: what problem commonly encountered with spreadsheets it is intended to solve
- a description of how to use it
- a list of wrinkles, that is, sometimes technically subtle details that can prove snags to trap those not familiar with them
- What you could do if you didn't have OAK. To identify credibly the capabilities that are truly innovative and unique to OAK, we have sought to be open about those commands that are to be found in most products of the genre, or perform functions that can be done in Excel, possibly with a little work.

2.2 Map

INTRODUCTION

The larger a worksheet becomes, the more difficult it is to manage, and finding particular cells can prove difficult in the same way as finding a specific location in a vast landscape.

The solution to both problems is to have a good map; one which produces a compact and accurate representation of a larger reality. One of the drawbacks of Excel is that it has no built-in facility for generating such a map.

The OAK Review | Workbook | Map command generates a graphical map of the structure of every sheet in a workbook or of a selection of the worksheets in a workbook, helping you to identify various worksheet features and to locate errors in the logic of the worksheets.

OAK WORKBOOK/WORKSHEET MAP

The OAK Review | Workbook | Map command provides you with a detailed map of the active workbook or a selection of worksheets within that workbook. It uses text, color, symbols, highlighting, and outlining to draw attention to various features on the worksheet.
WHY WOULD YOU WANT TO USE IT?

Map Workbook/Worksheets is primarily used as an auditing tool to allow worksheets to be rapidly checked for errors and inconsistencies. By highlighting the different worksheet features using different cell formats, you can easily understand the logic of the worksheet and find many mistakes.

By creating a new map (and saving it) each time you make modifications to your workbook, you create an audit trail of the development of the workbook.

When printed, a worksheet map can be used to easily explain the overall structure and logic of the worksheet to somebody else.

HOW TO USE IT

Choose the Map command from the OAK Review ribbon tab. The Map Workbook/Worksheets dialog box is displayed:
There are five groups of options on the dialog box. The first three (Formulas, Constants and Other), allow you to choose which of these types of cells to add to the map. The Output option specifies where to put the maps that are generated and the Input options specify whether to generate maps of every worksheet in a workbook or of a selection of worksheets from a workbook.

OAK has a set of default options. If you change the options on the dialog box, your choices will be retained until OAK is closed; you can press the ‘Default’ button to return to the pre-specified settings.

FORMULAS

Map Formulas. Checking this box will add cells containing formulas to the map. The cells to be mapped can be refined by using the remainder of the options in the group; unchecking this checkbox disables the remainder of the options in the group.

- Numbers. This checkbox allows you to refine the selection of formula cells to be added to the map. All cells with formulas that evaluate to numbers, for example “=Price*Volume” or “=4+5”, will be added to the map.

- Logicals. This checkbox allows you to refine the selection of formula cells to be added to the map. All cells with formulas that evaluate to the logical values TRUE or FALSE, for example “=Revenue>0” or “=4<5”, will be added to the map.

- Text. This checkbox allows you to refine the selection of formula cells to be added to the map. All cells with formulas that evaluate to a text string, for example “=IF(Revenue>Cost,”Saving”,”Borrowing”)”, will be added to the map.
- Error. This checkbox allows you to refine the selection of formula cells to be added to the map. All cells with formulas that evaluate to an error, for example “=VALUE!*SalesUnit” or “=7/0”, will be added to the map.

The relationship between formulas in neighboring cells is determined and represented in the map by the addition of symbols to the cells; these are explained in the Legend section.

CONSTANTS

Map Constants. Checking this box will add cells containing constants to the map. The cells to be mapped can be refined by using the remainder of the options in the group; unchecking this checkbox disables the remainder of the options in the group.

- Numbers. This checkbox allows you to refine the selection of constant cells to be added to the map. All constant cells that contain numbers, for example “500” or “10%”, will be added to the map.

- Logicals. This checkbox allows you to refine the selection of constant cells to be added to the map. All constant cells which contain the logicals “TRUE” or “FALSE” will be added to the map.

- Text. This checkbox allows you to refine the selection of constant cells to be added to the map. All constant cells which contain text, for example “Year ending”, will be added to the map.

- Error. This checkbox allows you to refine the selection of constant cells to be added to the map. All constant cells which contain an error value, for example “#REF!” or “#VALUE!”, will be added to the map.

- Copy Values. Use this checkbox to copy the values of constant cells into the map. By default this option is checked because it helps you to navigate around the map. If it is unchecked, numeric values are indicated by #; text values by X, logical values by L, and error values by Err.

OTHER

Map Named Ranges. Checking this box will outline the areas which belong to named ranges. Note that the names “Print_Area” and “Print_Titles” are mapped using a different color in order to avoid possible confusion due to overlapping named ranges.

Map Arrays. Checking this box will crosshatch the areas which are part of an array formula or a data table.

OUTPUT

This group selects whether the legend is generated and offers a radio button selection of one of three options for the location where the maps will be placed:

- New Workbook. Selecting this radio button will create the maps in a new workbook. This is the default setting.
Using OAK: OAK commands

- Same Workbook. Selecting this radio button will create the maps on new worksheets in the active workbook. Each map will be placed next to the worksheet to which it refers.

- Overlay. Selecting this option will overlay the maps on the existing worksheets. Warning: use this option with care as this function changes the original model. Always save your file before using this option.

- Show Legend. Use this checkbox to generate the Worksheet Map Legend on a separate worksheet as part of the map making process.

INPUT

These options select which worksheets should be mapped.

- Whole Workbook. Choose this option to generate a map of every worksheet in a single open workbook.

- Selected Worksheets. This allows a map to be generated for each one of a selection of worksheets from a single workbook.

LEGEND

The legend can be generated as part of the mapping process by checking the relevant box on the dialog. The legend can also be found in the online help.

The chart below identifies the symbols used in the worksheet maps. A color version of this legend can be found on the inside back cover of this manual.
To make maximum use of the power of the Worksheet Map it is important to be able to interpret it correctly. This requires knowledge of the symbols used and an understanding of how they are related to each other.

The map symbols have been divided up into groups by type: formula cells, constant cells, and other features.

- **Formula cells:** Cells which contain formulas are represented in this example by purple shading although the coloring of each consistency direction can be configured individually. Symbols are used to indicate the relationship between formulas in neighboring cells.

  - **Distinct formula cell (purple):** This cell contains a formula which is inconsistent with the formula in the cell to its left and in the cell above. The red triangle in the top right corner indicates a cell note into which the formula has been copied; hovering the cursor over the cell reveals the formula. Note that if the formula is very long, the cell note contains an error message instead.

  - **Formula consistent with cell to left (purple):** This cell contains a formula which is consistent with the formula in the cell to its left.

  - **Formula consistent with cell above (purple):** This cell contains a formula which is consistent with the formula in the cell above.

  - **Formula consistent with cell above and cell to left:** This cell contains a formula which is consistent with the formula both in the cell to its left and in the cell above.

- **Constant cells:** Constants can be divided into 3 distinct groups; numbers, text, and logicals. Each is represented by a different color in the map.

The values can be copied to the map... ...or can be replaced by a code symbol.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Number constant" /></td>
<td>This cell contains a numerical value. The actual value from the cell has been copied into the map.</td>
</tr>
<tr>
<td><img src="image" alt="Number constant" /></td>
<td>This cell contains a numerical value. The actual value from the cell has been copied into the map, but the cell is too narrow to display the value.</td>
</tr>
<tr>
<td><img src="image" alt="Text constant" /></td>
<td>This cell contains a text value. The actual text from the cell has been copied into the map.</td>
</tr>
<tr>
<td><img src="image" alt="Text constant" /></td>
<td>This cell contains a text value. The actual text from the cell has been copied into the map, but the cell is too narrow to display the text.</td>
</tr>
<tr>
<td><img src="image" alt="Logical constant" /></td>
<td>This cell contains a logical value. The actual value from the cell is not included.</td>
</tr>
<tr>
<td><img src="image" alt="Logical constant" /></td>
<td>This cell contains a logical TRUE value.</td>
</tr>
<tr>
<td><img src="image" alt="Logical constant" /></td>
<td>This cell contains a logical FALSE value.</td>
</tr>
</tbody>
</table>
Using OAK: OAK commands

- Other: These formats represent cells with additional properties. The error format overwrites those for formulas and constants while the other formats overlay those for formulas and constants.

<table>
<thead>
<tr>
<th>Error cell</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error cells indicate an error cell. This could either be the result of a formula or a constant. These cells should always be investigated and the error corrected.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Named range area</th>
</tr>
</thead>
<tbody>
<tr>
<td>This indicates the area covered by a named range.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Array area</th>
</tr>
</thead>
<tbody>
<tr>
<td>This indicates the cells which contain an array formula.</td>
</tr>
</tbody>
</table>

WRINKLES

This command will not function if applied to a protected worksheet. You must first manually unprotect it.

The colors used in the map can be altered with OAK’s Options command.

Excel cannot handle a worksheet in which every cell is a different color, typeface and font size. Excel keeps a list of all the different combinations of cell formats in use in a workbook, and beyond some limit will allow no further permutations of formatting to be added because the list has a fixed size and has become full. Though there is no danger of hitting this limit when drawing a map in a new workbook, OAK may take Excel beyond this limit when the map is superimposed on a worksheet that is already busy in formatting terms, since OAK achieves the mapping effect by applying appropriate formatting to cells.

The result will be that OAK will deliver an incomplete map. It will issue a warning that it has had to stop the action prematurely. Operis is working on addressing this. The workarounds are to

- remove some of the formatting from the workbook before doing the OAK mapping
- superimpose the maps one worksheet at a time until the limit is reached

WHAT YOU COULD DO IF YOU DIDN'T HAVE OAK

Nearly all Excel add-ins that are similar in purpose to OAK have a feature resembling OAK’s map workbook/worksheets tool, something that

- shows how each cell is related to neighboring cells
- highlights named ranges by drawing borders around them.
- identifies error cells.

Distinctive qualities of OAK’s version of this function are that
the maps it produces seek to provide little visual clutter that can distract from the objective of seeing which cells are inputs, what type of inputs they are, and which cells are formulas.

- cells that are part of arrays are also highlighted

If you didn't have OAK or any other add-in, you would have to rely on features built in to Excel to get as close as possible to the same insights.

- To learn where named ranges lie found, set the zoom in Excel to 25%. Do this by going to View | Zoom and selecting the 25% button. At this zoom level Excel will draw borders around the named ranges.

- To find any error cells on a worksheet, select Home | Editing | Find & Select | Go To Special and then choose the formula and errors options. Excel will select all of the error cells on the worksheet and you can move between them by pressing the tab key, or perhaps color them distinctively.

- A similar method will find constants and formulas and arrays.

- You can display all the formulas in a worksheet by using Formulas | Formula Auditing | \showformulas{} (shortcut Ctrl+`, an obscure key that is in the top left corner of US and UK keyboards, between the F1, 1 and Tab keys). But Excel won't show you which formulas are distinct and which merely copies of neighbors.

### 2.3 Compare | Workbooks

One of the most common problems encountered by spreadsheet users is that, over time, they forget what they have changed in successive versions of the same workbook. The problem is even more pronounced if several people have worked on the same workbook.

As each worksheet grows in size and complexity, and as workbooks accumulate more and more worksheets, it becomes increasingly difficult, if not impossible, to determine how the workbook as a whole has been modified.

To ask the question more directly: how can you tell if the workbook you are using today is exactly the same as the one you were using last week or a month ago?

It might take days or weeks to perform the tedious task of manually comparing the formulas or values on one worksheet with those on another. Even then, you would not be able to prove to someone else that you had not made a mistake while performing the comparison.

Fortunately, there is an alternative: the Compare Workbooks/ Worksheets tool. This tool can determine the differences between two worksheets or two workbooks. This comparison is typically completed within minutes, saving a significant amount of time and effort. The same comparison can also be repeated by somebody else.
Using OAK: OAK commands

Once the discrepancies have been identified and noted, a sister tool, the Remove Compare Modifications tool, can then be used to remove any alterations that were made to either workbook or worksheet during the comparison process.

Note: The Compare Workbooks command compares the contents of the cells on the respective worksheets in two workbooks. It does not compare the cell formats, cell comments, charts, modules, worksheet objects, or VBA projects in those two workbooks.

OAK WORKBOOK/WORKSHEET COMPARISON

The Compare Workbooks/Worksheets command actually links together several different functions. At the simplest level, it allows you to align two worksheets that are similar in overall vertical layout.

At its most exhaustive, it performs a worksheet-by-worksheet comparison of the contents of all the cells in two workbooks. It finds the appropriate pair of worksheets to compare, aligns them, compares them cell by cell, highlights their differences, and produces an output report detailing these differences.
Three of the options making up this command (Highlight Differences On Worksheet; Align Rows; and Align Columns) will alter your worksheets. Though there is an option to Remove compare modifications, you would be wise to keep a copy.

WHY WOULD YOU WANT TO USE IT?

This command is useful whenever you need to find the differences between two workbooks or two worksheets. This issue could arise in a number of ways.

- You have made several changes to your workbook and saved it as a new version. After a week you realize that one of the changes you have made has resulted in unexpected side-effects. But in order to figure out which one, you need to know all of the differences between the current and previous versions.
- You are part of a team building an Excel model. One of the other team members is out of the office for the day and you need to ascertain how the model was altered yesterday in order to decide what you need to do today.
- Having completed the final version of a workbook, you suddenly realize you need to go back and document the version history, describing the modifications in each of the last 3 versions.
- You have finished building a model and are running a simple sensitivity analysis. Although you have a good idea about which outputs should change when you alter specific inputs, you want to be absolutely sure you don't miss anything. This is a good way to test the logical flow of your workbook.
- You have just received a text file containing an updated report of the detailed costs of initiating a business plan. You would like to quickly compare these figures with the report you were emailed last week, which is in the same format.

In each of these cases, Compare Workbooks/ Worksheets provides a quick and powerful method for performing the comparison between the two relevant worksheets or workbooks.

HOW TO USE IT

When the Compare command is chosen from the OAK Review | Compare menu, the dialog box shown below will appear. It contains:

- two workbook and worksheets selection areas on the left.
- three sets of comparison options on the right.
The first thing to decide is whether you would like to perform a workbook or worksheet comparison. If you need to compare all of the cells in two workbooks, follow these steps:

1. Select one of the workbooks from the list of currently open workbooks as Selection #1;
2. Select the other workbook from the list of currently open workbooks as Selection #2;
3. Check the 'Compare All' checkbox.
4. Choose the relevant comparison options;
5. Click the ‘Compare’ button.

This command first finds all of the worksheets in both workbooks that have the same name. The order of the worksheets within each workbook does not matter as long as the worksheets have identical names. It then compares, one-by-one, the cell contents of each pair of worksheets with the same names. Cell formats, cell notes (comments), charts, modules, worksheet objects, and VBA projects are not compared.

The command will align worksheets, highlight cell differences, and write a difference report for each pair of worksheets compared if these principal options were checked on the dialog box. In addition, a summary report, which provides an overview of all comparison results, will be generated.

You cannot compare a workbook with itself.
COMPARE WORKSHEETS

If you only need to compare the cells on two worksheets either within the same workbook or between two separate workbooks, you can do so by:

1. Uncheck the 'Compare All' checkbox.
2. Selecting the workbook that contains the worksheets from the 'Workbooks' list in Selection #1, and then selecting the appropriate worksheet from the 'Worksheets' list in Selection #1;
3. Selecting the same workbook in Selection #2 and the appropriate worksheet from the 'Worksheets' list in Selection #2;
4. Choosing the relevant comparison options;
5. Clicking the ‘Compare’ button.

If the appropriate principal comparison options have been checked, the comparison tool will align the two worksheets, highlight the cells which differ, and produce a difference report.

You cannot compare a worksheet with itself.

PRINCIPAL COMPARISON OPTIONS

The principal options that determine how to execute the cell comparison are: Write Report; Highlight Differences On Worksheet; Align Rows; Align Columns; Group Blocks of the Same Modification in Report; and (compare) Names. Each of these options is useful under different circumstances and is described in more detail below. Note that at least one of the first four options must be selected in order to perform any comparison exercise.

Highlight Differences. This option causes every cell, which differs between two worksheets, either by its location or content, to be compared and highlighted wherever a difference is identified. The command also has an auxiliary option, which enables you to specify the exact color in which differences would be shown.

To update your color preference, simply click on the Set Colors button located in the bottom right-hand corner of the Compare Workbook/Worksheet dialog box. A secondary dialog box, entitled Specify Comparison Colors, will open. This allows you to specify a unique color for each of the modification actions that OAK will make during the alignment and comparison processes. You can set the color for an action by clicking on the respective Set Color button and choosing a color from the available color palette or from the Custom colors library.

By default: pale blue is used to mark all cells that have been modified, deleted or added; pale pink is used to mark a cell that belongs to separate arrays between the two worksheets; while yellow signifies that a blank row has been inserted; and, green indicates that a new column has been inserted.
The Highlight Differences option enables you to instantly detect any discrepant cells between two workbooks/worksheets. It is especially useful when trying to distinguish between slightly modified versions of the same worksheet. The default (and recommended) option is to conduct all comparisons with this option turned on.

Align Columns. This option compares the horizontal alignment of the two worksheets in question before conducting the cell comparison. If two worksheets are found to be similar in layout from left-to-right but do not map directly to the same column numbers, the Align Columns command will attempt to align them.

If this option is checked in the Compare Worksheet/Workbook dialog box, upon clicking the Compare button, the Column Alignment dialog box displayed above will appear. At this point, you will be required to specify a maximum of four contiguous rows that would be used to test the horizontal alignment of the two worksheets being compared. As a guide, Operis recommends that the selected row or rows should contain timelines, as these are least likely to change between different iterations of the same workbook/worksheet.

In order to carry out the alignment, the Align Column tool will attempt to align the two worksheets through a series of column insertions and deletions. All inserted columns are easy to detect as they are highlighted in green, or in whichever color that you stipulate using the Specify Comparison Colors | Blank Alignment Column option described above.

Align Rows. This option compares the vertical alignment of the two worksheets in question before conducting the cell comparison. If two worksheets are found to have similar layouts from top-to-bottom but do not map directly to the same row numbers, the Align Rows command will attempt to align them.

If this option is checked in the Compare Worksheet/Workbook dialog box, upon clicking the Compare button, the Row Alignment dialog box displayed above will appear. At this point, you will be required to specify a maximum of four contiguous columns that would be used to test the vertical alignment of the two worksheets being compared. As a guide, Operis recommends that the Selected column or columns should contain text, as these are least likely to change between different iterations of the same workbook/worksheet.

In order to carry out the alignment, the Align Rows tool will attempt to align the two worksheets through a series of row insertions and deletions. All inserted rows are easy to detect as they are highlighted in yellow, or in whichever color you stipulate using the Specify Comparison Colors | Blank Alignment Row option described above.

The Align Row and Align Column tools are both very useful options because it is very rare to find perfectly aligned worksheets, ready for comparison. Attempts to bypass these options and directly compare two unaligned worksheets would result in numerous, incorrect cell discrepancies being “identified”, when in fact the cell value is the same but the row or column has just been shifted. Ideally, the Align Row option should always be checked, while the Align Column option serves as an additional check that the worksheets are properly aligned before any comparison begin.
Write Report. This option generates an output report containing a list of all the cells that are different between each pair of compared worksheets, along with the contents of the discrepant cells. The structure of the report is a single workbook that contains:

- Individual reports that list the changes between the individually compared worksheets; and
- A summary report, located at the end, which provides an overview of all the changes detected between the two workbooks/worksheets.

It is important to note that even if the Highlight Differences option is selected and the discrepant cells are evident on the worksheet, the Write Report option offers a convenient solution to the often necessary requirement of documenting all cell differences between two versions of the same worksheet. Furthermore, the individual reports contain hyperlinks that lead to the worksheets on which the discrepancies occur. In a large workbook, such functionality can be particularly beneficial for navigation.

The default setting is to generate a report at the end of the comparison process.

Group Blocks of the Same Modification. This option condenses the output report into a summary of only distinct discrepancies between the different worksheets. By selecting the Group Blocks of the Same Modification option, OAK will only report one difference for every distinct row or column discrepancy between both Worksheets.

Compare Names. Compares the names in the workbooks/worksheets and produces a report detailing the differences. This can be important when comparing formulas, because a pair of workbooks with exactly the same formula content could produce very different results if those formulas refer to names that have different values between the workbooks/worksheets.

TEST TYPE OPTIONS

When performing a workbook or worksheet comparison, you can choose to compare either the Formulas or the Values in the cells on those worksheets. The difference between these two options is the way in which cells containing formulas are compared.

Using the Formula itself. The default option is to compare the formulas on the two worksheets. This means that in those cells, which contain formulas, the formulas are compared with no regard for their resulting values. Thus, for a given pair of cells that are compared, if the formulas are the same, but the values are different, no difference is reported for that pair of cells. It should be noted, however, that when comparing cells that do not contain any formulas, the values would then be compared.

- Using the Calculated Value. When the Calculated Value option is chosen, the inverse is the case: the formulas within the cells are ignored and the comparisons are now made between the actual values, which the formulas evaluate to.
- Case Sensitive Comparison. This option allows you to “force” OAK to treat texts with different capitalization profiles as being different. The option can be used in conjunction with both Formula and Calculated Value comparisons. In the case of the former, OAK is
sensitive to differences in the capitalizations of the names used in the formula, while the latter is a proxy for testing that ordinary cell texts in both worksheets have the same capitalizations throughout.

REPORT FORMULA STYLE OPTIONS

When both the Write Report option and the Formulas test type option are selected, the differences between formulas can be reported in either one of two styles: A1 or R1C1. These two options simply alter the presentation of the output report but do not affect the results of the actual worksheet comparison.

Using the A1 style will cause discrepant formulas to be reported with cell references of the form C25 or D18; while the R1C1 style will report the same formula discrepancy using cell references of the form R[i]C[j], where i and j are integers other than zero.

The advantage of using A1 style is that the formulas are easier to understand. Using R1C1 style, however, is more compact because similar formulas are grouped together. For example, consider the following worksheet:

If we write the formulas using A1 style cell referencing, each formula is written differently with the middle formula simply written as: “=D4*(1+E3/100)”. Using R1C1 style cell references, however, all three formulas can be grouped together using the following notation: “=RC[-1]*(1+R[-1]C/100)”. This makes the output reports more compact but less transparent.

REPORT CHANGE TYPE

The reports for each sheet comparison contain a column labeled "Change Type" which shows the type of change that has been detected. These are as follows:

<table>
<thead>
<tr>
<th>Change Element</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>Constant value</td>
</tr>
<tr>
<td>F</td>
<td>Formula</td>
</tr>
<tr>
<td>-</td>
<td>Blank (NOT caused by insertion of alignment columns or rows)</td>
</tr>
<tr>
<td>A</td>
<td>Alignment blank (i.e. an empty cell inserted as part of an alignment column or row)</td>
</tr>
<tr>
<td>{F}</td>
<td>Array formula</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Change Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>CF</td>
<td>Constant to formula</td>
</tr>
</tbody>
</table>
CC      Constant to different constant
FC      Formula to constant
FF      Formula to different formula
F{F}    Formula to array formula
{F}F    Array formula to formula
{F}{F}  Array formula to different array formula (including changes in array dimensions)
{F}C    Array formula to constant
C{F}    Constant to array formula
-C      Inserted constant
C-      Deleted constant
-F      Inserted formula
F-      Deleted formula
-{F}    Inserted array formula
{F}-    Deleted array formula
CA      Constant to alignment blank
AC      Alignment blank to constant
FA      Formula to alignment blank
AF      Alignment blank to formula
A{F}    Alignment blank to array formula
{F}A    Array formula to alignment blank

WRINKLES

The Compare Workbook/Worksheet command will not succeed if one of the worksheets is protected. You must first unprotect the protected worksheet before running the command.
As mentioned earlier, the Compare Workbooks/Worksheets command only compares the contents of the cells on the respective worksheets, and if requested, the names; it ignores the presence of cell notes (comments), chart sheets, macro modules, and objects (e.g., charts, drop downs etc.) embedded on worksheets. It is possible for two workbooks/worksheets that are reported as having no differences to be different in terms of these items.

Two workbooks can appear to be identical but give very different results as a result of containing names that are differently defined. OAK will not detect such differences. If requested, it will compare the names in a workbook.

Excel cannot handle a worksheet in which every cell is a different color, typeface and font size. Excel keeps a list of all the different combinations of cell formats in use in a workbook, and beyond some limit will allow no further permutations of formatting to be added because the list has a fixed size and has become full. OAK may take Excel beyond this limit when it marks differences it identifies on a worksheet that is already busy in formatting terms, since OAK achieves the marking by applying appropriate formatting to cells.

The result will be that OAK will deliver an incomplete comparison. It will issue a warning that it has had to stop the action prematurely. Operis is working on addressing this. Possible workarounds are to

- remove some of the formatting from the workbook before doing the OAK comparison
- perform the comparison one worksheet at a time until the limit is reached
- turn off the option to mark the difference found on the original worksheets, and to learn what has changed by referring to the separate comparison reports instead
- use Excel 2007-2016, in which the format limits have been raised from 4,000 to 64k per workbook.

PRINCIPAL COMPARISON OPTIONS

The principal comparison options are almost completely independent of each other. That is to say that, you can perform a comparison with only one of the options being selected, with the exception of the "Group blocks of the same modification" option, which is dependent on the Write Report option. Otherwise, any combination of the options is possible.

For example, you could simply select the Align Rows option to only align two worksheets without actually performing a comparison. You could also just have the different cells highlighted without aligning the sheets or producing an output report.

However, it is Operis’s experience that, in most cases, checking the Align Rows, Align Columns and Highlight Differences options gives the best results.
ALIGN ROWS AND ALIGN COLUMN TOOLS

The Align Rows and Align Columns tools eliminate the need for you to manually verify the alignment of the worksheets before performing a comparison. However, it should be noted that their ability to align worksheets is limited by the structure and layout of the worksheets.

If your worksheets are well-structured and consistently use one or two designated columns for descriptive text, then the Align Rows tool will perform optimally, while the Align Columns tool would work best when a consistent timeline has been adopted for both worksheets being compared. Therefore, we recommend that you designate one or two columns in your worksheets (columns A and B for example) as text descriptions for each row and use them consistently. Likewise, two or more rows at the top of each worksheet can be designated for timelines.

If there is simply not enough information in the alignment columns or rows to re-align the worksheets, the both the Align Row and the Align Column tools will fail, with an error message informing you that the worksheets could not be aligned.

Similarly, if the same text or date label appears several times in the same alignment column or row, both tools may be unable to complete the alignment process because there will be no certainty as to which rows or columns should be aligned.

The Align Rows (and Align Columns) tool can also fail to align worksheets, where the order of two blocks of rows (or columns) has changed. This would come about by deleting a block of rows (or columns) from one section of the worksheet and inserting it somewhere else on the sheet. The Align Rows (or Align Columns) tool will not be able to align the sheets because it does not attempt to change the row order on the sheets. In this case, you would have to manually change the row order before the command can function properly.

Note that one way around the issue of re-ordered rows is to sort the two worksheets to be compared by the same column. To do this, highlight the entire block of rows and columns which need to be sorted and choose Excel’s Data | Sort command. While we have had success using this procedure ourselves, we recommend that you use this option with care and always save your files before proceeding to do this.

OAK also offers advanced alignment options. To activate these, check the "Use Advanced Alignment Options" check box, and click the ellipsis button to its right. This activates the advanced alignment form.
This enables the user to specify that rows and columns be aligned using different data to that which is compared. Rows and columns can be aligned by formula, by value, or simply inherit whatever option is used for comparison. Likewise with case sensitivity, which can be inherited from the compare options, or specified explicitly.

TEST TYPE OPTIONS

When comparing worksheets containing no formulas (only numbers), it makes no difference whether the Formulas or Values test type option is chosen for the comparison. Furthermore, the Formula style option is only effective when both the Write Report option and the Formulas test type option are jointly selected.

COMPARING TEXT FILES

Finally, we made the claim at the beginning of this chapter that the Compare Workbooks/Worksheets command could be used to compare reports that were in text file format. In fact, it can be used to compare any two non-Excel files which Excel can recognize.

It is possible, for example, to open a text file in Excel by simply choosing Office/File button | Open, changing the ‘Files of type’ field to ‘Text Files’, and choosing a file from the list. The Text Import Wizard dialog box is then displayed and you must decide how to import the file.

Having done so, the text in the text file will be arranged in cells on the current worksheet. You then open another file and perform the same operation with the file you wish to compare the first one with. Once set up, run the Compare Workbooks/Worksheets command to find the differences between them.

If you are having trouble with the rows in the two worksheets being in the wrong order, try using the Data | Sort command to sort the data blocks on the two worksheets first. This can often help to make the Align Rows and Align Columns tools function correctly.
WHAT YOU COULD DO IF YOU DIDN'T HAVE OAK

Excel and some packages that compete with OAK offer similar means by which two worksheets can be compared; however, only a few possess the ability to re-align the worksheets.

The benefits to be gained by solely comparing two worksheets are limited, as there is no guarantee that the cells being compared are intended to contain similar information, or if they represent separate blocks of data. To get around this problem, re-alignment of the two worksheets is necessary as it forces like-for-like comparison, which will yield more meaningful results from the comparison exercise.

2.5 Compare | Ranges

In addition to comparing workbooks and worksheets, OAK will also compare ranges of the same size and shape.

As blocks of formulas are copied, pasted and modified throughout the workbook, it becomes useful to know if two blocks of cells are the same.

To ask the question more directly: how can you tell if the range at location 1 contains the same formulas that are contained in the matching cells in the range at location 2?

It might take 30 seconds or even several minutes to perform the tedious task of manually comparing the formulas or values in one range with those in another. Even then, you would not be able to prove to someone else that you had not made a mistake while performing the comparison.

Fortunately, there is an alternative: the Compare Ranges tool. This tool can determine the differences between two ranges. This comparison is typically completed within seconds, saving a reasonable amount of time and effort. The same comparison can also be repeated by somebody else.
Using OAK: OAK commands

For ease of identifying which ranges have been compared, the Compare Ranges tool will color the selected ranges according to the Range Selection color specified in the Compare Color Options.

Once the discrepancies have been identified and noted, a sibling tool, the Remove Compare Modifications tool, can then be used to remove any alterations that were made to the affected worksheets during the comparison process.

Note: The Compare Ranges command compares the contents of the cells in the respective ranges in the selection. It does not compare the cell formats or comments in those ranges.

OAK RANGE COMPARISON

One of the options making up this command, Highlight Differences On Worksheet, will alter your worksheets. Though there is an option to Remove compare modifications, you would be wise to keep a copy.

WHY WOULD YOU WANT TO USE IT?

This command is useful whenever you need to find the differences between two blocks of cells.
HOW TO USE IT

When the OAK Review | Compare | Ranges command is chosen the dialog box shown below will appear. It contains:

- two range specification areas on the left
- two sets of comparison options on the right

COMPARE RANGES

The first thing to decide is the ranges you want to compare. OAK will attempt to initialize the Compare Range options dialog using the current selection.

1. If desired, equalize the range dimensions using the ‘=’ buttons.
2. If desired, select different ranges using the ‘...’ buttons.
3. Choose the relevant comparison options.
4. Click the ‘OK’ button.

This command compares, one-by-one, the cell contents of each range. Cell formats, cell notes (comments) and embedded objects are not compared.

The command will write a difference report for the pair of ranges compared if that option was checked on the dialog box.
PRINCIPAL COMPARISON OPTIONS

There are 3 principal options that determine how to execute the cell comparison. These are: Write Report; Highlight Differences; and Group Block of the Same Modification. Each of these options is useful under different circumstances and is described in more detail below. Note that at least one of the first two options must be selected in order to perform any comparison exercise.

Highlight Differences. This option causes every cell, which differs between two worksheets, either by its location or content, to be compared and highlighted wherever a difference is identified. The command also has an auxiliary option, which enables you to specify the exact color in which differences would be shown.

To update your color preference, simply click on the Set Colors button located in dialog box. A secondary dialog box, entitled 'Comparison Colors', will open. This allows you to specify a unique color for each of the modification actions that OAK will make during the comparison processes. You can set the color for an action by clicking on the respective Set Color button and choosing a color from the available color palette or from the Custom colors library.

The Highlight Differences option enables you to instantly detect any discrepant cells between two ranges. It is especially useful when trying to distinguish between slightly modified copies of the same range. The default (and recommended) option is to conduct all comparisons with this option turned on.

Write Report. This option generates an output report containing a list of all the cells that are different between the compared ranges, along with the contents of the discrepant cells. The structure of the report is a single workbook that contains an overview of all the changes detected between the two ranges.

It is important to note that even if the Highlight Differences option is selected and the discrepant cells are evident on the worksheet, the Write Report option offers a convenient solution to the often necessary requirement of documenting all cell differences between two versions of the same worksheet. Furthermore, the individual reports contain hyperlinks that lead to the worksheets on which the discrepancies occur. In a large workbook, such functionality can be particularly beneficial for navigation.

The default setting is to generate a report at the end of the comparison process.

Group Blocks of the Same Modification. One of the newest options incorporated into the Compare tool suite, is the option to condense the output report into a summary of only distinct discrepancies between the different ranges. By selecting the Group Blocks of the Same Modification option, OAK will only report one difference for every distinct row or column discrepancy between both ranges.
TEST TYPE OPTIONS

When performing a comparison, you can choose to compare either the Formulas or the Values in the cells in those ranges. The difference between these two options is the way in which cells containing formulas are compared.

Using the Formula itself. The default option is to compare the R1C1-style formulas in the two ranges. This means that in those cells, which contain formulas, the formulas are compared with no regard for their resulting values. Thus, for a given pair of cells that are compared, if the formulas are the same, but the values are different, no difference is reported for that pair of cells. It should be noted, however, that when comparing cells that do not contain any formulas, the values would then be compared.

- Using the Calculated Value. When the Calculated Value option is chosen, the inverse is the case: the formulas within the cells are ignored and the comparisons are now made between the actual values, which the formulas evaluate to.

- Case Sensitive Comparison. This option allows you to “force” OAK to treat texts with different capitalization profiles as being different. The option can be used in conjunction with both Formula and Calculated Value comparisons. In the case of the former, OAK is sensitive to differences in the capitalizations of the names used in the formula, while the latter is a proxy for testing that ordinary cell texts in both ranges have the same capitalizations throughout.

REPORT FORMULA STYLE OPTIONS

When both the Write Report option and the Formulas test type option are selected, the differences between formulas can be reported in either one of two styles: A1 or R1C1. These two options simply alter the presentation of the output report but do not affect the results of the actual worksheet comparison.

Using the A1 style will cause discrepant formulas to be reported with cell references of the form C25 or D18; while the R1C1 style will report the same formula discrepancy using cell references of the form R[i]C[j], where i and j are integers other than zero.

The advantage of using A1 style is that the formulas are easier to understand. Using R1C1 style, however, is more compact because similar formulas are grouped together. For example, consider the following worksheet:

If we write the formulas using A1 style cell referencing, each formula is written differently with the middle formula simply written as: “=D4*(1+E3/100)”. Using R1C1 style cell references, however, all three formulas can be grouped together using the following notation: “=RC[-1]*(1+R[-1]C/100)”. This makes the output reports more compact but less transparent.
WRINKLES

The Compare Ranges command will not succeed if one of the worksheets is protected. You must first unprotect the protected worksheet before running the command.

As mentioned earlier, the Compare Ranges command only compares the contents of the cells on the respective ranges; it ignores the presence of cell notes (comments), and embedded objects. It is possible for two ranges that are reported as having no differences to be different in terms of these items.

Users should be aware of the effect of Excel's format limitations on compare operations.

TEST TYPE OPTIONS

When comparing worksheets containing no formulas (only numbers), it makes no difference whether the Formulas or Values test type option is chosen for the comparison. Furthermore, the Formula style option is only effective when both the Write Report option and the Formulas test type option are jointly selected.

WHAT YOU COULD DO IF YOU DIDN'T HAVE OAK

Aside from using another product, a comparison of values could be performed using conditional formatting, or a block of TRUE/FALSE formulas equal to the difference between the compared cells.

2.6 Compare | Remove Compare Modifications

The OAK Review | Compare | Unmodify removes any changes to a spreadsheet that have been made by the OAK Review | Compare | Compare command, putting the spreadsheet back to the condition it was in before it was compared.

WHY WOULD YOU WANT TO USE IT?

The OAK Review | Compare | Workbooks/Worksheets/Ranges command inserts rows and columns into the spreadsheets being compared in order to minimize the number of reported differences. That action alters the spreadsheet. Under some circumstances it is desirable to reverse the alterations.

Operis makes extensive use of this facility when commissioned to audit financial models. It refers to the client's spreadsheet as the MUT, the Model Under Test. It builds a new spreadsheet, which it calls the AWP, or Audit Working Papers, which contain hundreds of tests. Some are obvious, like whether any balance sheet balances. Others are more proprietary.

Frequently the client provides a revised MUT, reflecting changes to the deal, or reactions to issues raised in the audit process. This MUT2 is highly likely to have a row layout slightly different from the original MUT1.
In such circumstances Operis will

- load into Excel the AWP linked to MUT1, and MUT2

- use OAK’s Compare feature to get the rows in MUT1 and MUT2 as aligned as possible

- use Excel’s Edit Links command to switch the links of the AWP from MUT1 to MUT2

- use OAK’s Remove compare modifications so that MUT1 and MUT2 are restored to their original conditions
The result is that the AWP extracts necessary information from the appropriate regions of MUT2 even though they are not necessarily in exactly the same position as they were in MUT1.

In this way, Operis can validate new versions of spreadsheets in a matter of minutes, which is handy if a deal is to be signed imminently. Spreadsheet auditors who do not have a process similar to this one are forced to impose a cutoff point beyond which they no longer accept changes, since they need time to redo all their work from first principles.

HOW TO USE IT

To remove the comparison modifications, select the OAK Review | Compare | Unmodify command and the following dialog box appears:

Select either ‘Whole Workbooks’ or ‘Selected Worksheets’ as necessary. In the corresponding boxes below each option, select the required workbook(s) or worksheets (Ctrl and click to select multiple items) by clicking on the names in the relevant box. Multiple worksheets can only be selected if only one workbook is selected.
WRINKLES

If, as well as aligning the spreadsheets, you have OAK deliver a report of the differences it found during its comparison, the removal of the aligning rows will leave the report referring to where the items used to be before the row removal, not where they are now.

WHAT YOU COULD DO IF YOU DIDN'T HAVE OAK

The question of how you would achieve a similar action if you did not have OAK is somewhat hypothetical. If you didn't have OAK, you would not have any changes made by OAK's Compare commands to reverse. (Unless you had an old version of OAK, which did have the Compare command, but not the means to remove the compare modifications.)

But leaving that point aside, OAK aligns spreadsheets by inserting rows which it marks clearly, by color and by tagging them with a comment. In the absence of the Remove compare modifications command, one could search for and delete the modifications made manually.

If you do not have links between the workbooks being compared, such as the ones to the AWP above, you don't need OAK to get back to the unaltered spreadsheet. Just discard it and reload it.

2.7 Names | Redefine names

Excel 2007 introduced a Names Manager which provides the functions performed by OAK Development | Names | Redefine and will perform them considerably faster. The command is valuable with earlier versions of Excel and provided also for consistency with earlier versions of OAK.

The OAK Development | Names | Redefine command intentionally resembles the Formulas | Defined names | Define name command built into Excel, but it enhances it in two respects.

- it allows you to change names that you have misspelled or otherwise wish to alter
- it allows you to define local names more easily.

WHY WOULD YOU WANT TO USE IT?

To rename a named range

From time to time you will find that you have used a name in a workbook and wish you’d called it something else. You might:

- misspell the name;
- want to change it so that it is consistent with other names that are related to it;
- realize that it just doesn’t accurately describe the area it relates to.
To alter the name, you need to

- delete the name you don’t like;
- define a new one that you prefer;
- alter all the formulas that use the old name, so that they instead use the new one.

The last step is quite a chore. There may be hundreds of formulas in the workbook that use the name, and they have all got to be identified and altered; these formulas may be widely scattered over the worksheets in the workbook. Furthermore, these formulas may even be referencing from different workbooks.

The OAK Development | Names | Redefine command carries out these three steps in a single operation, updating cell references across worksheets and workbooks.

To define a local name

Excel makes it rather difficult to define local names; you have to use the Formulas | Defined names | Define name command and type the name that you want along with a sheet prefix using the correct syntax.

The OAK Development | Names | Redefine dialog has a simple check box that specifies the nature (global or local) of the name that should be added. You can also define a name as local by simply giving it a name beginning with an exclamation point (!). This is true for both the ReDefine Name and ReCreate Names commands. This has the added benefit that any name beginning with “!” is immediately recognizable as a local name.

Note: Even though Excel does allow you to create both a local name and a global name on the same worksheet with the same name, we strongly recommend that you not do this.
HOW TO USE IT

Adding a name
1 select the range of cells you wish to name
2 enter the range you would like to name (or click the button on the right and select the range);
3 select OAK Development | Names | Redefine;
4 type the name you want in the New Name box;
5 if you would like the new name to be a local name, tick the Local Name checkbox;
6 press the ‘Add’ button.

Renaming a name
1 select OAK Development | Names | Redefine;
2 select the name that you wish to rename;
3 type the name you want in the New Name box;
4 press the ‘Rename’ button.

Alternatively, you can:
1 select the range of cells you wish to rename;
2 enter the range you would like to name (or click the button on the right and select the range);
3 select OAK Development | Names | Redefine;
4 type the name you want in the New Name box;
5 press the ‘Rename’ button.

Deleting a name
1 select OAK Development | Names | Redefine;
2 select the name that you wish to delete;
3 press the ‘Delete’ button.

WRINKLES

This command has been designed to mimic the behaviour of the equivalent command built into Excel. In particular, it can be used to define new names, as well as re-define existing ones, so it can be used as a complete, more capable replacement for the built in command.

You cannot use this command to define a new name, or redefine an already existing name, on a protected worksheet. If you redefine an already existing name while there are formulas on protected worksheets which reference that name, the pre-existing name in the
formula will not be replaced. Previous versions of OAK would not allow any change to existing names referenced by protected worksheets at all.

NAMING CONVENTION

There are certain guidelines which you should adhere to when constructing your naming convention; the following rules define what constitutes a valid name in OAK:

- names must contain at least 4 letters so that they can be distinguished from references to columns;
- names must begin with either a letter or an exclamation point (!); remember that any name beginning with “!” will be defined or created as local;
- names cannot contain punctuation characters or spaces apart from underscores (_) or full stops (.)

If you violate any of these rules, an error dialog will be displayed.

LIMITATIONS

You cannot use this command to rename a name to a name that already exists. You will receive a warning dialog if you attempt to do so.

When renaming a local name, the sheet reference at the start of the name must be retained in the new name to ensure the name is still defined as a local name, otherwise it will be defined as a global name.

You cannot use this command to redefine a local name as a global name or a global name as a local name in a single step. In other words, the local name checkbox only applies when defining a new name. In order to redefine the scope of a name, you must first delete the name in question and then define the new name. However, this still leaves you with the task of manually updating all references to that name by either adding or deleting the worksheet name reference which comes before the actual name.

WHAT YOU COULD DO IF YOU DIDN'T HAVE OAK

As mentioned above, Excel 2007 introduced facilities for adjusting names and defining local names easily. In earlier versions of Excel, redefining a name can be done without OAK but is a tedious process for large worksheets.

The name can be deleted and re-created by using Formulas | Defined names | Define name selecting the misapplied name and clicking Delete, and then renaming the range by selecting it and using Formulas | Defined names | Define name, typing in the correct name and clicking Add. The formulas which need to reference the changed name then need to be found and changed to match with the new name.

Excel’s Find and Replace (Ctrl+H) command may be able to speed the process up slightly, where you could select the block of formula that needs to be changed, pressing
Ctrl-H, typing the old name as the Find what criteria and the new name as the Replace with criteria.

## 2.8 Names | Recreate names

The OAK Development | Names | Recreate command intentionally resembles the Formulas | Defined Names | Create from selection command built into Excel, but it enhances it in two respects.

- it allows you to change names that you have misspelled or otherwise wish to alter
- it allows you to define local names more easily.

### WHY WOULD YOU WANT TO USE IT?

**To rename a named range**

From time to time you will find that you have used a name in a workbook and wish you’d called it something else. You might:

- misspell the name;
- want to change it so that it is consistent with other names that are related to it;
- realize that it just doesn’t accurately describe the area it relates to.

To alter the name, you need to

- delete the name you don’t like;
- define a new one that you prefer, which may mean adjusting a label adjacent to the previously defined range, selecting that range, and reusing the Formulas | Defined Names | Create from selection command
- alter all the formulas that use the old name, so that they instead use the new one.

The last step is quite a chore. There may be hundreds of formulas in the workbook that use the name, and they have all got to be identified and altered; these formulas may be widely scattered over the worksheets in the workbook. Furthermore, these formulas may even be referencing from different workbooks.

The OAK Development | Names | Recreate command carries out these three steps in a single operation, updating cell references across worksheets and workbooks.

**To define a local name**

Excel's Formulas | Defined Names | Create from selection command can make many names at one go, but they are always global names, associated with a workbook. It can’t be used to make local names which have the narrower scope of a worksheet.
HOW TO USE IT

The ReCreate Names command can be used to create many single-row or single-column names in one operation. However, it takes a moment to ensure you have the correct structure in place. Labels that match the names that you wish to create must be placed in a single column or row on the worksheet adjacent to the ranges you wish to define.

If the labels are in a row, the individual columns in the selection are named according to the label that corresponds to each column. If the labels are in a column, the individual rows in the selection are named according to the label that corresponds to each row.

If an individual row or column within the selection is already named, this command will rename that row or column to match the corresponding label.

Follow these steps to create a new name or to rename an existing name:

1. If there isn't already an existing label in a cell next to the range you wish to name, type the name you wish to create in a cell adjacent to the range you wish to name;
2. Select the range you wish to name including the row or column labels;
3. Select OAK Development | Names | Recreate;
4. Designate the location that contains the labels in the ReCreate Names dialog;
5. Select to which extent you want OAK to update the name references;
6. Press the ‘OK’ button.

A name created by this procedure does not include the cells that contain the labels.

Any name beginning with an exclamation point (!) will be created as local to that worksheet. The dialog box also allows for local names to be created by ticking the Force Local option.

WRINKLES

This command has been designed to mimic the behaviour of the equivalent command built into Excel. In particular, it can be used to create new names, as well as re-create existing ones, so it can be used as a complete, more capable replacement for the built in command.

You cannot use this command to define a new name, or redefine an already existing name, on a protected worksheet. If you redefine an already existing name while there are formulas on protected worksheets which reference that name, the pre-existing name in the formula will not be replaced. Previous versions of OAK would not allow any change to existing names referenced by protected worksheets at all.

The OAK ReCreate Names dialog box differs from the one in Excel in one small detail. Each of the choices presented, identifying a locations in which to create the name or names (for example ‘Top Row’, ‘Left Column’, etc.) is exclusive of the others. In Excel,
however, this is not the case, and you can check more than one of the choices at the same time.

In Excel, if you use the Formulas | Defined Names | Create from selection command to create a block of names and then immediately follow it with the Formulas | Defined names | Defined name | Apply Names command, those names which were just created will be highlighted in the list of names. This particular feature is not replicated in OAK. However, the OAK Development | Names | Apply command makes it easy to apply many names at a time.

To perform the renaming, OAK has to identify all the relevant formulas that make use of one or more of the chosen names, and rewrite them one by one. This can be time consuming.

Excel 2007 introduced the possibility of altering a name and have all mentions of that name update instantly. The facility is offered through the Excel user interface in the form of the Names Manager, which provides renaming capabilities similar to those long offered by OAK. OAK Development | Names | Recreate is implemented in a way that uses this new capability, so long as the option is selected to apply to the whole workspace. Narrower scopes (active workbook, active worksheet or current selection only) still have to be done by OAK by altering formulas one at a time, because there is no way of limiting the scope of the action built in to Excel.

WHAT YOU COULD DO IF YOU DIDN'T HAVE OAK

Excel 2007 introduced facilities for adjusting names and defining local names easily. In earlier versions of Excel, redefining a name can be done without OAK but is a tedious process for large worksheets.

If it was Excel's Formulas | Defined Names | Create from selection command that was used to create the original version of the name, remember to alter the adjacent label from which Excel took the name so that it is consistent with the changed name.

2.9 Names | Apply names

Both OAK and Excel offer Apply names commands. Both versions of the command search through a spreadsheet for opportunities to replace coordinate references in formulas with names that have been defined by the user.

There is a lesson on the use of the Excel version of the command in Lesson E of the tutorial on names in this help.

The OAK version of the command is closely modeled on the Excel one, but has one very small convenience.
Using OAK: OAK commands

In early versions of Excel, it was possible to select a portion of the list of names offered by the Apply Names dialog by choosing one name, then holding down the Shift key while using the cursor up and down keys to extend the selection to neighboring entries.

A particular case of this facility was that it was possible to select the entire list of names with two keystrokes, Home (to get to the top of the list) and Shift+End to get to the bottom of the list, selecting as you go.

OAK restores this convenient facility which was lost when the way Excel handles list boxes was altered with release 5 of the program. In that and all subsequent versions of Excel, selecting 20 names from the list requires 20 clicks of the mouse, or 20 presses of the space bar and 19 presses of an up or down arrow key.

WHY YOU MIGHT WANT TO USE IT

To enthusiasts of using names in spreadsheets, selecting a single cell (Apply names follows the One-Cell rule) and applying every one the names that have so far been defined is a common and useful action. Excel's dialog manager makes this impossibly slow. OAK permits this in a few quick keystrokes and allows it to be performed whenever a few more names have been defined in a worksheet.

WHAT YOU COULD DO IF YOU DIDN'T HAVE OAK

As noted, the OAK command is closely modeled on the Excel version and merely extends it. All the actions it performs are achievable through the Excel user interface, but with considerable work.

2.10 Names | Deapply names

OAK's DeApply Names command is used to completely or partially eliminate the use of names in formulas by replacing them with the actual cell coordinates that they reference. It is the inverse of OAK's Apply Names command.

WHY WOULD YOU WANT TO USE IT?

Opinion is divided between those who like the use of names in Excel formulas and those who don't. OAK's Deapply names allows enthusiasts for names to use them when building models, and then to strip them out from any copy sent to recipients who have an aversion to the things.

Model auditors who use code inspection often wish to satisfy themselves that every name in a worksheet refers to the area it is intended to refer to. OAK can help in this process by building a database of the names that need checking. If there are hundreds of names defined, checking them one at a time can be a burdensome process. One way of removing the step altogether is to have OAK simply remove the names from the spreadsheet formulas.
One drawback of using names in spreadsheets is that it is awkward to change your mind about which names to use late in the development process. For example, if you have used Sales throughout a large model, and realize that you would prefer to have used Turnover, you have to alter every instance of the word Sales in every formula the mentions it. Excel 2007 has addressed this issue, and OAK provides the ReDefine Name and ReCreate Names commands for Excel 2003. But a further option is to Deapply the existing names, getting the formulas back to coordinate notions, and then to create new names and apply them in the normal way.

HOW TO USE IT

With the workbook containing the names-based formulas open:

- Select the OAK Development | Names | Deapply command. This will open a dialog box similar to the one shown below.

![OAK Deapply Names Dialog Box](image)

- The dialog box shows you all the names within the workbook and allows you to select all or part of these to be removed from the formulas.
- Use the Ignore Constants checkbox to avoid deapplying names that refer to constant values rather than ranges.
- Once you have selected the names you want removed, you have the option of specifying which areas of the workbook the DeApply command should be run over. The three options are: a selected region within the active worksheet, the entire worksheet or the whole workbook. Note that in order for the selected region option to be available, the selection must be made before selecting the DeApply name command. Once you are satisfied with the names you want to remove and where you want these changes to be effected, click the DeApply button.

Tip: Particularly if you are removing all the formulas from a spreadsheet that uses a lot of them, OAK will change many or most of the formulas in your model. It is only sensible to make a copy of your workbook before using the Names | Deapply command. Compare the key results between the versions to make absolutely sure that the alterations haven't altered the effect of the spreadsheet.
HOW IT WORKS

To deapply names, OAK is actually understanding the formulas algebraically, identifying all uses of names, extracting and substituting the definitions of the names, and inferring which cells are involved in the calculation after row and column matching has taken effect. In practical terms, the function is implemented as an in-place reconstruction of the original formula.

WHAT YOU COULD DO IF YOU DIDN'T HAVE OAK

So far as Operis is aware, there is no capability equivalent to OAK's Deapply Names command in any competing Excel add-in.

There is a way to remove names from a formula manually.
1 Highlight a cell that contains a formula using one or more names
2 Double click on one of the names in the formula.
3 Press F5. This is the short cut for Go To; it will bring up a dialog box in which the coordinate equivalent of the name is highlighted
4 Press Ctrl+C to copy the coordinates on to the clipboard
5 Press Escape to dismiss the Go To dialog. You will return to the formula that contains the unwanted names, still being edited.
6 Press Ctrl+V to paste the coordinates into the edit box. It will overtype the unwanted name.
7 Repeat as required for any other names in the formula.
8 Press Enter to accept the now name-free formula

This is too manual a process to perform on any scale; does only part of the job because it doesn't address issues of column and row matching; and does nothing useful with names that are themselves defined in terms of other names.

WRINKLES

The DeApply names command does not work when the worksheet or workbook under modification is protected. You must first manually unprotect the worksheet/workbook before running the command.

DeApply names can be instructed to attend to a selected region within the active worksheet; the entire worksheet or the whole workbook. In the first of these cases, it obeys the One Cell Rule.

While the DeApply command removes the names from the formulas, it does not delete them from the selection/worksheet/workbook. As a result, using of Excel, or OAK's Apply names command should exactly reverse the effect of Deapplying the names, putting the
names back as they were in the first place. If the names are no longer wanted, they can be removed manually.

If there are many names in a worksheet, and OAK is asked to Deapply many of them, it can take a long time. The action is making a considerable change to the spreadsheet, potentially rewriting nearly all of the formulas. If you are certain that a worksheet is left to right consistent (that is, all formulas after the first on each row are copies of that first one) then it may be quicker to deapply names in the first column of formulas and then copy the result into subsequent columns.

The technology used to Deapply names is ambitious. While it has been in use for several years and tested carefully, we would certainly recommend that a version of the spreadsheet is kept as it was before attempting the Deapply, and retained until one is satisfied that the process has worked as intended and the spreadsheet continues to give the same results.

### 2.11 Names | Remove #REF! names

The OAK Development | Names | Remove #REF! command builds a list of all the names in the active workbook that have the text #REF! in their definition. These names are displayed in a dialog and you can then delete all the #REF! names in a single step.

**WHY WOULD YOU WANT TO USE IT?**

If you give a name to a cell which you later delete, the formulas which refer to these names will now display #REF! By looking under ‘Refers to’ in the Define Name dialog box, Formulas | Defined names | Define name, we are able to see what exactly has been deleted to cause the #REF! error:

- if ‘Refers to’ displays something like “=#REF!$A$5”, it indicates that the cell disappeared because you deleted the whole sheet that it was on;
- if ‘Refers to’ displays something like “=Sheet1!#REF!”, it indicates that the worksheet is still there, but the cell has disappeared, most likely because you removed the row or column containing it.

Names with #REF! in their definitions should be removed from your model, as they refer to cells that are no longer in the workbook. At the very least, they clutter up the list of names, where they can’t do anything useful. At worst, they may actually still be being used in a formula, leading it to give results different from those intended.

**HOW TO USE IT**

The Remove #REF! Names command displays a list of all the names that have the string “#REF!” anywhere in their definition. If you would like to permanently remove all of these names from your workbook, simply select all names by using the ‘Select All’ checkbox, then click the ‘Remove Selected’ button. If you would only like to remove certain names,
then select them as required and click the ‘Remove Selected’ button. Otherwise, press
the ‘Cancel’ button.

You can use this command to:

· check that there are no such broken names in your model;
· remove any that are in the model with a single button;
· if you prefer, go and check out the names identified in the list one by one and fix them
  individually.

WHAT YOU COULD DO IF YOU DIDN'T HAVE OAK

If you didn't have OAK, the only ways to remove broken names from a worksheet are

· to delete them manually one at a time using the Define name dialog;
· to write a few lines of Visual Basic that will achieve the same thing.

WRINKLES

Very exceptionally, a name can have the string “#REF!” in its definition intentionally; OAK
will not know that and will include the name in the list to remove.

Removing #REF! names will likely result in any formulas which refer to the deleted names
to return #NAME? errors as, now, the names that are referred to are undefined instead of
referring to a cell or range that no longer exists. Further attention to is needed to remove
these #NAME? errors.

2.12 Names | Delete names in cells

The Delete Names in Cells command provides a way of rapidly deleting a quantity of
names that you don’t want any more or have made by accident.

WHY WOULD YOU WANT TO USE IT?

Excel offers two ways to define names in a workbook.

· The built-in Excel Formulas | Defined names | Define name command allows you to add
  names to a workbook, and to delete them; but it only allows you to do so one name at a
time.
· The built-in Excel Formulas | Defined Names | Create from selection command allows
  you to add a whole block of names to a workbook, many at a time; but it offers no
  facilities for deleting them.

Operis finds the second method the most convenient way to specify the names in a
workbook. It makes a model self-documenting because it leaves an indication of the
names that have been defined next to the values that are being referred to; and it allows lots of names to be defined in a single action.

But there is a dangerous aspect to Formulas | Defined Names | Create from selection: if you use this command by mistake, or mis-select the area that is meant to contain the names you are creating, you can add a great quantity of unintended names to the workbook.

Under such circumstances OAK Development | Names | Delete command can come to the rescue. Select the cells containing the labels that match the names you want to delete, select this command, and the unwanted names will all disappear.

HOW TO USE IT

To use this command, follow these steps:
1. Select a range containing labels that match the names you wish to delete;
2. Select OAK Development | Names | Delete;
3. A warning dialog is displayed, where you must press ‘OK’ to continue; remember that this command has no Undo facility.

A given name may appear several times in a workbook, once at book level (a global name) and once on individual worksheets (a local name). In these circumstances OAK is designed to give the same result as would arise if the names were typed, one by one, into the Excel Formulas | Defined names | Define name command and the ‘Delete’ button pressed. In other words:

- the Delete Names command will never delete local names not defined on the active worksheet;
- if the text in a selected cell corresponds to both a global name and a local name defined on another worksheet, the global name is deleted;
- if the text in a selected cell corresponds to both a global name and a local name defined on the active worksheet, that local name is deleted. Note that if the name appears twice in the list, the global name will be deleted as well.

What you could do if you didn't have OAK

You don't need OAK to get rid of unwanted names. You can delete them through Excel, using its Formulas | Defined names | Define name command. However, you will need to handle each name individually, and to pick out the names to be removed from a potentially long alphabetic list.

WRINKLES

This command does not work when applied to protected worksheets. You must manually unprotected them first.
This command does not follow the One-Cell rule. This is to prevent all the names that happen to be mentioned on a worksheet from being deleted whenever the command is selected with only one cell selected.

If you want to delete all the names mentioned on a worksheet, select the whole sheet (Ctrl+A does this quickly) before using the command.

2.13 Names | Localize names in cells

Excel supports two kinds of names, ones that belong to the whole workbook (global names) and ones that live on individual worksheets (local names).

The OAK Development | Names | Localize command turns existing global names into local names.

WHY WOULD YOU WANT TO USE IT?

The fastest way to add a quantity of names to a workbook is to use the Excel Formulas | Defined Names | Create from selection command. However, this command creates global names; sometimes you want local names instead.

Using the OAK Development | Names | Localize command immediately after the Excel Formulas | Defined Names | Create from selection command (with the selection undisturbed) will cause the global names just created to be transformed into names that are local to the active worksheet.

HOW TO USE IT

To use this command, follow these steps:

1. Select a range containing labels that match the names you wish to localize;
2. Select OAK Development | Names | Localize;
3. A warning dialog is displayed, where you must press ‘OK’ to continue; remember that this command has no Undo facility.

Any global name that refers to the active worksheet and that is identified by a label in one of the cells will be deleted from the workbook and a replacement for it will be created locally on the active worksheet.

HOW IT WORKS

OAK looks at all the cells in the current selection. Where a cell contains text that coincides with a defined name, the defined name is deleted and a new one that is of narrower scope (a local, sheet level name) is created with the same definition.
WRINKLES

This command does not work when applied to protected worksheets. You must manually unprotect them first.

This command does not follow the One-Cell rule. This is to prevent all the names that happen to be mentioned on a worksheet from being localized whenever the command is executed with only one cell selected. If you want to localize all the names mentioned on a worksheet, select the whole sheet (Ctrl+A does this quickly) before using the command.

Once a name has been localized, any references to it on other worksheets need to be qualified with the sheet name. That needs to be done manually; OAK doesn’t do it automatically.

WHAT YOU COULD DO IF YOU DIDN’T HAVE OAK

It is possible to mimic OAK’s action by deleting a global name and then defining a new local name to replace it. Excel’s standard Formulas | Defined names | Define name command is all that is necessary to do it. The process is fine for a handful of names but becomes very slow if many names must be converted.

2.14 Names | Build name database

The Excel Formula | Defined names | Use in formula | Paste names command has an option, Paste List, which allows you to place on a worksheet a list of the names that have been defined in a workbook.

OAK takes this idea and radically expands on it. In addition to displaying the title and definitions of the names in a workbook, the OAK Build Name Database provides information on the properties of the names. These increase the chances of finding mistakes in them.

WHY WOULD YOU WANT TO USE IT?

A database of names provides the means to spot a variety of mistakes that are not otherwise easily detected. The benefits of this functionality are listed as follows:

- The names database shows all of the names in a workbook at once; in contrast, Paste List will only show the global names and those local names that are local to the currently active worksheet.

- The names database allows you to see that local names that are intended to be repeated on several worksheets do in fact appear consistently on all of them.

- If you create a name on one worksheet that already exists on another, the new name will not replace the old one. There will instead be two names in the workbook, one global and one local to the worksheet on which it is located. These ghost names show up in the name database.
**Using OAK: OAK commands**

- Names do not have to refer to specific cell references or ranges; instead they can refer to constants, relative ranges or formulas. These names are flagged in the name database.

- If you have named the entries in a 20-year profit and loss statement, then the names should each be 20 columns wide. If some are 21 and others 19, you may have highlighted one column too many or too few when you defined them. The width of each named range is shown in the name database.

- It is possible for two or more names to be defined so that they refer to the same area. Normally this is a mistake, resulting from forgetting that something has already been given a name and naming it a second time. The duplicated names show up in the “overlap report” if that option of the name database is checked.

**OUTPUT**

The Names Database collects the following information about every name in the active workbook:

- **Full Name.** The name, including the sheet specifiers for local names (this is in the hidden column A);

- **Short Name.** This is the same as the long name, except that sheet specifiers are not included for names;

- **Defined on Sheet.** This lists the sheet that the name is defined on. This is blank for global names and is simply the sheet specifier for local names;

- **Formula.** The range, constant or formula that the name refers to (this is blank for names that refer to relative ranges);

- **Refers to Sheet.** Specifies the sheet that the name refers to (flags in this column indicate constant, formula, relative, and error names, as well as those linked to another workbook); and,

- **Address.** The absolute address that the name refers to.

The following additional information is also included, however these are most useful for names that refer to ranges:

- **Areas:** The number of independent, non-contiguous ranges that the name refers to;

- **Top:** The topmost row of the range that the name refers to;

- **Bottom:** Bottommost row of the range that the name refers to;

- **Height:** The number of rows in the range that the name refers to;

- **Left:** The leftmost column of the range that the name refers to;

- **Right:** The rightmost column of the range that the name refers to; and

- **Width:** The number of columns in the range that the range refers to.

- **Hidden:** The hidden state of the name.
The Data | Sort & Filter | Filter option is, by default, activated on the output worksheet. This enables you to easily filter the Names Database for a specific criterion in every column. This is especially useful when your workbook contains a large number of names.

**HOW TO USE IT**

Open a workbook, then select the OAK Review | Names | Build name database command.

A dialog box similar to the one shown above will be displayed. From this, you can specify the following options:

- produce a report listing overlapping named ranges;
- exclude special names from the overlap test;
- place the resulting reports either on a new worksheet in the active workbook, or in a new workbook; and,
- check for hidden names.

**FINDING OVERLAPPING RANGE NAMES**

If you wish to produce a report listing overlapping named ranges, it is worth noting that some names, particularly “Print_Area” and “Print_Titles”, will tend to lead to a great number of spurious overlaps. These names are required for print formatting and, as such, do not alter the functionality of the model. As they will most likely cover a significant area of any worksheet on which they are defined, these names can be expected to overlap with many of the other names that are used for calculations.

To prevent such “deliberate” name overlaps from being reported, you should choose the “Exclude Special Names from Overlap Test” option. Any additional names, which will result in overlap fails that you deem to be inconsequential, can also be excluded from the Overlap Test by simply adding it to the list of Special Names using the Add button on the Name Database Options dialog box. Likewise, if you decide that a name should no
Using OAK: OAK commands

If a name is no longer be exempt from the Overlap Test, then this name can be deleted from the Special Names list by using the Remove button.

The results of the Overlap Test are compiled into a separate worksheet, next to the Names Database. This report shows the identity and range of the intersecting names, as well as the cell in which they overlap.

This is a powerful tool for identifying obsolete names still present in the workbook that should be deleted.

CHECK FOR HIDDEN NAMES

One of the options available to you when building up the Names Database is the decision of whether to include hidden names in the database or to omit them completely.

It should be noted that the hidden names that this option deals with are those that have been hidden from the worksheet user at the macro-level i.e. by altering the “Visibility” property of the name from within visual basic. To clarify, the Check for hidden names option does not describe a check for names that occur in hidden cells on the active worksheet or hidden worksheets. Such names will always be detected, irrespective of whether the Check for hidden names option is selected or not.

WRINKLES

If two names refer to overlapping ranges, they appear twice in an overlap report, once in one sequence, the other time in the reverse order.

WHAT YOU COULD DO IF YOU DIDN'T HAVE OAK

As alluded to at the beginning of this section, Excel’s Paste List option offers the fundamental name storing capabilities similar to those provided by OAK’s Names Database. However, the difference lies in the richness of the information. The information provided about each name when using the Excel Paste List option is restricted to the identity of the name and its range location. By comparison, OAK’s Name Database option provides that information, in addition to information regarding the range location, size and type.

2.15 Formula | Reconstruct

The OAK Review | Formula | Reconstruct command causes OAK to generate a new workbook which contains one or more worksheets. Each worksheet sets out a report which seeks to make it easy to understand how a calculation derives a result.
The formula \( =A1000+B2000+C3000 \) is hard to understand because the cells referred to are far apart on a worksheet. The formula

\[ =\text{Sheet1!A}1000+\text{Sheet2!B}2000+\text{Sheet3!C}3000 \]

is even harder to understand because the cell referred to are widely separated in three dimensions. A reconstruction of a calculation shows what it would look like if those cells that play no part in the calculation were removed from the spreadsheet, so that those cells that are involved could be repositioned near to the formula.

The ability to reconstruct calculations is one of the most distinctive and technically ambitious capabilities of OAK and is explained in detail in the section of this help that sets out Basic concepts in OAK.

When the command is invoked, OAK checks that the selection is suitable for reconstruction.

- A single area
- All or part of one row

If it is not suitable, OAK presents a suitably descriptive message of complaint.

If it is suitable, OAK presents a dialog box.
Using OAK: OAK commands

The address of the selection is shown in field **Row to reconstruct** at the top of the dialog. If you realize that is not the range you want to reconstruct, press Cancel to dismiss the dialog, change the selection and try OAK Review | Formula | Reconstruct again.

Since a well-written calculation will be built up in several cells, each with a formula calculating a piece of the result, OAK attempts to show the entirety of the calculation by combining the formula in a cell with the formulas in the cells it mentions. This action can be controlled by setting the **Maximum number of levels** of precedent formula substitution. A low number will quickly produce a reconstruction that may describe only part of a calculation. A high number will take longer to produce a reconstruction that may have a long formula that is hard to read. A happy medium may lie between these extremes. As an aid to finding that midpoint, OAK can be instructed to **Show all levels up to limit**, in which case it will produce a workbook with worksheets showing levels of formula expansion from zero (no formula expansion: only the selected formula is reconstructed) up the limit specified.

When **Confine to additions** is switched on, OAK will combine the formula in a cell with the formulas in the cells it mentions, up to the number of levels of precedents specified, but only to the extent that the result uses the operators +, - and SUM. This is useful for generating the components of a grand total without any intervening subtotals.

OAK’s pursuit of insight into the whole of a calculation by combining formulas can produce results that are long and complicated. Two options are offered to address this.

- **OAK can be instructed to Optimize** the formula that it develops in reconstructing a calculation, that is, to try to simplify it without changing its effect. This can be valuable when the reconstructed formula becomes very long. The action is the same as that performed by OAK’s Formula | Optimize command.

- **OAK can be instructed that a reconstruction should be Confine[d] to active path.** It will reduce conditional and lookup functions to their results, eliminating the need to include in the reconstruction branches and values that are not activated by current inputs. The action is the same as that performed by OAK’s Formula | Prune command. The various relevant functions can be selected or deselected for pruning individually.

Several controls on the dialog influence how a reconstruction report is displayed.

- **OAK can be instructed to give an indication of where in a worksheet the cells that are mentioned in a formula are located, by showing additional Rows of context either side of rows containing cells that participate in a formula.** The context rows are shaded grey so that they appear fainter, to signal that they do not participate in the calculation, but are there for information.

- **OAK can be instructed to reproduce any timeline or similar dimensional information by reproducing the frozen rows of the window in which the reconstructed worksheet is displayed on the screen, or the print titles set for printing the worksheet.** Since these resemble the rows of context, in aiding interpretation of the formula, but not participating in it, they too are shaded grey.
• OAK will freeze a specified number of columns at the left of the reconstruction sheet so that they remain visible when the rest of the spreadsheet is scrolled.

2.16  Formula | Optimize

OAK’s Formula | Optimize command applies expression simplification to the formula in the active cell. If it finds that the formula can be expressed more simply, it offers the option to replace the original formula with the simplified one.

![Image of OAK's Formula | Optimize dialog box showing original and optimized formulas]

OAK is equipped with about fifty algebraic rules which it uses for transformations such as these.

If none of these rules matches any portion of the formula in the active cell, OAK will say so.

![Image of OAK's Information dialog box showing no optimizations found]

Tip: OAK can perform as valuable as service in not offering an optimization for a formula, as in delivering one. For example, OAK would not offer any change to IF(A1+B2+C3+D5>5+E6+F7+G8+H9,A1+B2+C3+D4,E6+F7+G8+H9), though it looks as if it should; the first and third expressions are not quite the same. Is it is mistake? Who knows. But it is certainly a candidate for careful checking.
WRINKLES

Some simplifying actions can be performed on formulas completely safely; that is, the simplified formula will always give the same answer as the original formula. For example, =A1++B1 can be rewritten as =A1+B1, and depended on to give the same answer.

But some transformations are not so safe. They will give the same answer most of the time, but not always. This point is explained further at What's safe in simplification.

OAK therefore divides the rules for algebraic transformation between

- ones that are Conservative, that is, can be applied in any circumstances without any danger of changing the meaning of a formula,
- ones that are Aggressive, that is, need to be applied carefully, because there are circumstances in which they might change the meaning of a formula.

Although OAK invites you to confirm that you would like it to make any formula changes that it proposes, there is no facility for undoing the change after it is made. Keep a copy of your spreadsheet.

2.17 Formula | Prune inactive path

OAK's Formula | Prune inactive path command causes OAK to rewrite a formula so that it easy to understand how its result is derived. It does this by removing branches in conditional functions that are not active given current inputs.

For example, the expression IF(A1>B2,C3,D4) evaluates to either C3 or D4, depending on the result of the A1>B2. Replacing the function with simply C3 or C4 leaves the spreadsheet giving the same answer as before, given unchanged inputs, and makes the calculation simpler and easier to understand.

WHY WOULD YOU WANT TO USE IT?

Expression pares down a spreadsheet in order to understand the essence of a calculation. It is particularly valuable in spreadsheets that make intensive use of OFFSET and INDIRECT; pruning will replace those terms with references expressed in simple coordinate notation which are very much easier to verify.

HOW TO USE IT

1 Select the area containing cells that are to be pruned.
2 Invoke the OAK Review | Formula | Prune command
3 A dialog will appear, listing the functions OAK is capable of pruning.
4 Select the wanted functions and press OK.
5 OAK will prune those formulas within the selection that are amenable to it.

WHAT YOU COULD DO IF YOU DIDN'T HAVE OAK

The nearest thing to OAK’s Formula | Prune command is the Formulas | Formula auditing | Evaluate formula command, which provides some facilities for probing a formula and understanding how it derives a result.

WRINKLES

Though the change that OAK makes to a spreadsheet leaves it delivering unchanged answers, that is only true so long as the inputs do not change. Different assumptions might result in different paths being taken through conditional functions and different values being returned from lookup functions. Under such conditions a pruned formula will give an answer that is simply wrong compared with the unaltered spreadsheet.

For this reason, it is essential that the OAK user keeps a copy of any spreadsheet that he or she applies OAK’s pruning to. The purpose of the pruning is to deliver a short lived spreadsheet that will be discarded after it has delivered insight into a particular calculation.

To be sure that OAK discards the right conditional paths and selects the wanted lookup values, ensure that the spreadsheet has been thoroughly recalculated before using this command.

Unlike many other commands in Excel and OAK, Formula | Prune does not follow the One-Cell rule, to prevent significant and possibly time consuming unintended changes to an entire worksheet when only one cell is selected. To prune all the cells in a worksheet, select the whole worksheet first.

2.18 Formula | Analyze Discrepancies

OAK’s Analyze Discrepancies command generates a new workbook containing a comparison of the derivation of two selected cells.

OAK seeks to highlight values that are common to the left and right halves of the report, and so contribute to each of the two cells being compared. It identifies like cells by color, and hyperlinks them together.
WHY YOU MIGHT WANT TO USE IT

Operis believes strongly that the most powerful way to verify a spreadsheet calculation is to demonstrate that the same answers can be derived using a completely different method. Developing a parallel reconstruction of key calculations is a central part of its approach to reviewing or auditing spreadsheet financial models.

It is not uncommon for a parallel reconstruction to give an answer that is different from the original model. If this did not happen at least occasionally, there would be no point in spending the time developing the reconstruction. In such cases, the question is: which model is correct? Is the model under test defective, or is our own reconstruction wrong? Often, both are wrong, which is even more time consuming to unravel.

OAK aims to accelerate the identification of the causes of discrepancies such as this. It generates a new workbook, with the two cells being compared in the centre, along with the difference between the two values. In the column to the left from the first cell being compared, and right from the second cell, are links to the cells that those formulas mention. Working left and right, there are further links to those cells' precedents. In this way the report presents a fan-out of the precedent hierarchy of each of the two cells.

HOW TO USE IT

1. Select two cells. (If they are not next to each other, click on one, then hold Ctrl down and click on the other to select a multi-area range.)

2. Invoke the OAK Review | Formula | Analyze Discrepancies command.

3. OAK will present a dialog.
4 Click OK to generate a report.

Most of the controls on the dialog are devoted to preventing the fan-out from becoming so large that the analysis takes to long to prepare. They provide for

- limiting the number of levels of formula precedent that OAK inspects: the maximum depth field at the top of the dialog

- instructing OAK not to follow paths that are not presently active: the prune inactive paths option in the center of the dialog. This is particularly helpful for making sense of formulas that use lookup functions such as VLOOKUP and HLOOKUP, or use the (dangerous, in Operis’s opinion) OFFSET or INDIRECT

- not looking at the precedents of ranges that have very many cells, such as =SUM(A1:Z1000), which has 26,000 of them: the maximum precedent cells field in the center of the dialog.

OAK considers references common to the left and right halves of the report, and so contributing to each of the two cells being compared, only if they originate from exactly the same cell on a worksheet. The options at the bottom of the dialog instruct OAK to apply the less stringent test of considering values to be equal if they are close in value, even if they are taken from different cells.
WRINKLES

Further detail of the ideas behind discrepancy analysis are to be found in the section of the help that discusses OAK Concepts.

When the command is invoked, OAK will complain if the selection is not suitable for discrepancy analysis, that is, it does not consist of two cells.

If the resulting discrepancy analysis has very few cells on it, perhaps as few as the original two cells and the discrepancy between them only, check whether the option is “Suppress display of all children with nodes without matches” turned on. If it is, try running the discrepancy analysis again with it turned off.

The discrepancy analysis has a number of other uses.

1. Rather than comparing a calculation with a reconstruction of that calculation, it can be valuable to compare two neighboring cells in the same calculation, for example, Operating costs in 2011 and 2012. The fan-outs show which items are common to the two calculations and which are the ones that make them different.

2. Some models operate at a fine level of granularity in the start-up or construction phase, switching to a coarser resolution when operations have got underway; perhaps monthly yielding to semiannual. A discrepancy analysis can compare a figure picked from the monthly section with an equivalent in the semiannual section, such as Operating costs in 2010 and 2020, to see whether their derivations are consistent.

3. The fan-out can be a useful display when trying to understand what is behind a single cell. To generate one, click on the cell, and Ctrl-Click on a blank cell. That will satisfy OAK’s requirement that the analysis should compare two cells, but save half the time by only investigating a single value.

WHAT YOU COULD DO IF YOU DIDN'T HAVE OAK

Before OAK provided them, Operis analysts frequently built discrepancy analyses by hand. It's possible, but it takes a long time.

Sometimes the source of discrepancies can be identified by zeroing out selected inputs until the discrepancy disappears.

Excel has more different facilities than many people realise for investigating a cell's precedents. Of these, the Formulas | Formula auditing | Evaluate formula command is the closest to discrepancy analysis in presentation.

2.19 Search Overview

Excel provides two commands that help you to locate cells with specific attributes in a workbook.
- The Excel Home | Editing | Find & Select | Find command searches the selection for either cell formulas or cell values that contain a user specified text string. This command selects a single cell at a time and is made to step through those cells that meet the criterion by pressing the Find Next button.

- The Excel Home | Editing | Find & Select | Go To command allows you to jump directly either to a named range or to one of the most recently visited cells. However, if you press the ‘Special’ button, this command displays a further dialog that allows you to jump to those cells in the selection that belong to a subgroup of the available cell types. These subgroups include, among other things, cells which contain constants or cells which contain formulas; both of these subgroups can be refined to return any combination of number cells, text cells, logical cells and error cells. The Excel Home | Editing | Find & Select | Goto Special command selects all of the cells that meet the specified criteria at once; you can then step through those cells using the TAB key.

The OAK Development/Review | Search functions have been designed to extend the range of cell attributes that can be located. For example, it is very interesting to know which cells are not being referenced as these must be outputs from the model (or they are surplus to requirements and should be deleted). Alternatively, you might have a workbook that is full of error cells simply because a single error value has propagated through all dependent formulas; Excel provides no tool to help you locate the original error value- this feature is added by OAK.

The OAK commands that are used to locate cells with certain attributes are grouped together on a single menu.

These commands select all cells that have the required attributes at once; you can step through the cells using the TAB key. Alternatively, the cells can be colored by using the Home | Font | Paint-pot icon. This latter approach has the advantage that you can examine the individual cells without having to worry about losing the selection.

All of the OAK Development/Review | Search commands follow the One-Cell Rule. None of them work when applied to protected worksheets.

2.20 Search | Arrays

This command selects all cells within the current selection that form part of arrays, including data tables. Array formulas are a special kind of Excel formula, which are described in detail among Excel concepts in the Reference material.

WHY WOULD YOU WANT TO USE IT?

Arrays are among the most complicated formulas that a workbook can contain and as such should be examined and tested particularly thoroughly.

Data tables often constitute the primary output of a model and are hence of particular interest to you. They also cause Excel to iterate and can therefore slow down the calculation speed of your workbook. You can actually set the calculation option in Excel so
that it does not automatically recalculate data tables, which is another reason why you might want to inspect them carefully.

Arrays and data tables are protected by Excel so that it is impossible to modify a single cell within an array, or the body of a data table, in isolation. This is because all cells that constitute a particular array, or the body of a particular data table, must have a common formula. A consequence of this is that Excel will refuse to insert and delete rows and columns through arrays or data tables.

By contrast, OAK provides replacement insert rows / columns and delete rows / columns commands which will work through arrays and data tables. A user might wish to use the OAK Development/Review | Search | Arrays command to locate, and then examine, all arrays and data tables before carrying out such an insertion or deletion.

HOW TO USE IT

Either select the range of cells you suspect contains an array or select just a single cell to find all the arrays on the worksheet. Then choose the OAK Development/Review | Search | Arrays command from the menu; the command will either select those arrays, including data tables, that intersect the current selection or return a “No array cells found in selection” message.

WHAT YOU COULD DO IF YOU DIDN’T HAVE OAK

At first sight there seems to be an overlap between the functionality offered by OAK’s Search | Arrays command and the current Array option of the Go To | Special command built into Excel. But they are not quite the same.

To recap, the Excel Go To command will identify and select the whole of any array formula of which the current active cell forms a part. It can be reached by pressing F5, clicking on the Special button at the bottom of the dialog which then appears, and choosing Current array from the options that are offered. The shortcut is Ctrl+/. Regardless of how many cells are initially selected, Excel’s Go To command will only consider the active cell. Since that cell can only be part of one array formula, Excel will only highlight a single array (or report that the cell is not in fact part of an array). By contrast, the OAK Development/Review | Search | Arrays command follows the One-cell rule which leads it to consider potentially many cells in its search. It can locate potentially many arrays on the worksheet. In the extreme case where only one cell is selected, it will identify all the arrays on the worksheet.

The Excel Home | Editing | Find & Select | Goto Special command is particularly useful when there is knowledge of the existence of an array, and there is some initial indication of its location within the workbook. On the other hand, the OAK Development/Review | Search | Arrays command can be used to locate numerous arrays within a workbook, without the benefit of prior information about where they are located.
As it confines its attention to the selected cells, OAK will only identify those cells that both form part of an array formula and are within the initial selection. Some cells might form part of the same arrays but lie outside the initial selection. OAK will not highlight them. For this reason, it is common to use the OAK command first to locate a worksheet’s arrays, and then the Excel command after that to understand the exact extent of any of the individual arrays.

### 2.21 Search | Merged cells

**MERGED CELLS**

This command selects all the merged cells which intersect the current selection.

**WHY WOULD YOU WANT TO USE IT?**

There are many practical reasons to dislike merged cells.

- Highlight a rectangular range of cells that includes some merged cells, and the selection will automatically be enlarged so that it includes the merged area. It is impossible to select a smaller area.

- You can select an entire column or row by clicking on its header. But if the column or row intersects any merged cells, those cells get selected too. That can be tiresome; in particular, it stops the Insert rows and Insert columns commands from working (see Chapter 10 for a way around this).

- You can select an entire column or row with the keyboard: Ctrl+Space selects the column, Shift+Space the row. But if the column or row intersects any merged cells, those cells get selected too, along with the entire columns or rows that they in turn intersect. In most circumstances that renders the ability to select rows and columns with the keyboard unusable. This is particularly confusing if the merged cells happen to be hidden.

- Text that won't fit in a cell is usually able to spill out into its neighbors, so long as they are unoccupied. In a merged cell, however, the text is truncated.

- Unlike every other cell formatting option, the merging of cells is not simply a characteristic of a format style; it actually changes the regular grid structure of your spreadsheet.

A common use of the Excel Merge and Center command is to center heading text above a number of columns over which the heading applies. The OAK Development/Review | Search | Merged cells command allows you to find any merged cells that exist on your worksheet. Once identified, the merged cells can then be reverted back into individual cells and formatted as follows to achieve the same appearance of text within merged cells:

- open the Format Cells dialog box either by right-clicking the mouse on an active merged cell or by using Ctrl+O,E;
- select the Alignment Tab;
Using OAK: OAK commands

- change the Horizontal Text alignment option to “Center Across Selection”;
- uncheck the Merge cells option under Text control; and,
- click OK to close the dialog box.

HOW TO USE IT

Either select the range of cells you suspect to contain merged cells or select just a single cell to find all the merged cells on the worksheet. Then choose the OAK Development/Review | Search | Merged cells command; it will either select those merged cells that intersect the current selection or return a “No merged cells found in selection” message.

WRINKLES

If the active cell happens to be a merged cell, then the OAK Development/Review | Search | Merged cells command will only identify this cell, ignoring other merged cells within the worksheet. In this case, the One-Cell Rule does not apply because the active cell is being treated as a selection of more than one item as opposed to a single cell.

WHAT YOU COULD DO IF YOU DIDN'T HAVE OAK

There are no comparable tools within Excel capable of locating multiple merged cells in a worksheet. A probable solution would be to implement a search algorithm in a macro. The identified merged cells can then be split apart using the “Center Across Selection” Horizontal Text alignment formatting option described above.

2.22 Search | Primary error cells

This command selects those error cells in the current selection that do not in turn reference another error cell in the current selection. This command is most useful when the entire worksheet (or equivalently a single cell) is selected.

WHY WOULD YOU WANT TO USE IT?

A single error value is usually propagated throughout a workbook because any cells that reference that error value (either directly or indirectly) will typically result in errors. You will often need to examine a large number of error cells one at a time in order to identify the small number of cells that need to be modified. This operation can be very time consuming.

The OAK Development/Review | Search | Primary error cells command identifies those cells which contain the original error values and allows you to concentrate on fixing the source error cells rather than finding them.
HOW TO USE IT

Select the range to be searched and choose the command from the menu; the command will either select those cells within the selection that contain primary errors or return a “No primary error cells found in selection” message.

WRINKLES

If you select a small number of cells on a worksheet and then select the OAK Development/Review | Search | Primary error cells command, the resulting cells might not actually be primary error cells. This is because they might refer to error cells outside your selection. Therefore, in most cases, we recommend running this command on a whole worksheet.

However, you may find that even then, the cells which are found by the command do not constitute primary errors because they refer to error cells on a different worksheet in your workbook. In this case you will have to run the command on that worksheet to find the true primary error cells.

Another option is to use OAK’s reconstructor to trace back the precedents of the cells found by the Primary Error Cell command. The reconstructor will lay out on a single worksheet a calculation equivalent to the one being examined. The result could then be inspected by eye or one could use OAK’s primary error cell search on it.

The Primary Error Cells function might display a “No primary error cells found in selection” message when the selection clearly contains error cells. This occurs when the workbook is circular and the primary error cells are on the circular path. In this case, each error cell does reference another error cell. This is just one reason why users should avoid building circular models.

WHAT YOU COULD DO IF YOU DIDN'T HAVE OAK

Excel contains a function that is able to select error cells. This function can be found under Home | Editing | Find & Select | Goto Special and then by selecting the formula and error options. This function follows the one-cell rule (see page 17). However, unlike OAK’s Primary Error command, Excel’s function will select all error cells within the current selection. This means that the primary error cell must still be found by checking individual error cells.

Excel does however contain another mechanism for identifying primary error cells.

Excel's Formulas | Formula auditing | Error checking (unlabelled icon ) | Trace error command will draw red arrows on a worksheet to trace the source of an error. The difference between this and OAK’s Search | Primary error cells command is that it will only find one primary error, even if several are present; and selecting the error cells requires using the mouse to navigate the arrows drawn on the worksheet.
2.23  Search | Hardcoded constants

This command selects all formula cells within the current selection that contain a numeric element. Formula cells that contain numbers only as part of text strings, function names, and name labels are not selected by this command.

The option is given to ignore certain numeric elements as determined by the user. By default, numeric elements that have the values 0 and 1 are ignored as these are required in many Excel functions.

WHY WOULD YOU WANT TO USE IT?

Good practice in developing models dictates that all input values should be clearly stated on the face of a worksheet rather than buried within formulas. This command allows you to locate any cells that contain buried numeric values; in many cases, these numeric values should be split out onto the face of the worksheets.

An example of such a formula is =A1/365. It embeds in the formula the assumption that all years have 365 days. In some cases, the extra day in a leap year may make this assumption wrong by a material amount.

What is dangerous about hardcoded constants is that they may escape checking. A common step in building or reviewing a model is to cross-check the numerical assumptions used against any documentation provided. Both Excel and OAK provides mechanisms for identifying the numerical assumptions.

- Excel allows for the constants and formulas in a worksheet to be determined using the Home | Editing | Find & Select | Goto Special command.
- OAK includes an option for lists constants in a convenient report using the Summarize command, against which a check with the documentation can be easily performed.

Hardcoded constants, on the other hand, will be not appear in either Excel's selection or OAK's listing, as they are embedded in formulas. They may escape checking. The OAK Development | Search | Hardcoded Constants command provides a method by which these hardcoded constants in formulas may be found and checked.

HOW TO USE IT

Select a range and choose the command from the menu; a dialog is displayed that gives you the option not to treat certain numeric elements as hardcoded constants. The command will either select those formula cells within the current selection that contain a buried numeric input or return a “No hard coded constants found in selection” message.

WRINKLES

Many Excel functions require user specified numeric parameters that cannot really be considered to be model inputs. An example of this is the MATCH function where the third parameter specifies the match type. Cells containing functions of this type are typically
selected by this command even though they are not strictly hardcoded numeric inputs. The majority of these cells can be avoided by opting not to treat 0 and 1 values as hardcoded constants.

We recommend that you periodically opt to treat 0 and 1 values as hardcoded constants as these numeric values can have a profound effect on the logic of a model and should be examined carefully (it is not suggested that these 0 and 1 values should be split out onto the face of the worksheets).

We also recommend in general that other numeric values are not excluded from being treated as hardcoded constants before careful consideration and examination of the model.

**WHAT YOU COULD DO IF YOU DIDN'T HAVE OAK**

There is no automatic way to detect hardcoded constants inside formulas using Excel alone without OAK.

### 2.24 Search | Constant formula cells

This command selects all constant formula cells within the current selection. A constant formula does not reference any other cells; instead, for example, it reads “=10+40”. This type of structure might be used when you want to record the derivation of a particular input value.

**WHY WOULD YOU WANT TO USE IT?**

Checking a model involves verifying that the input assumptions are correct, and that the formulas do what is intended. Both Excel and OAK can help with this.

- Excel allows for the inputs and formulas in a workbook to be identified using the Home | Editing | Find & Select | Goto Special command
- OAK lists distinct inputs and formulas in a convenient report, using the Summarize Workbook command, against which a check with the documentation can be easily performed.

The risk with constant formula cells is that what are really inputs are listed as formulas, and so may escape checking. The Search | Constant Formula Cells command in OAK provides a method by which constant formula cells may be found and checked.

**HOW TO USE IT**

Select a range and choose the command from the menu; the command will either select those formula cells within the current selection that do not reference any other cells or return a “No constant formula cells found in selection” message.
WHAT YOU COULD DO IF YOU DIDN'T HAVE OAK

There is no automatic way to detect constant formula cells inside formulas using Excel alone without OAK.

The closest option is to use Excel’s Home | Editing | Find & Select | Goto Special | Formulas command. It allows the user the option to search for formulas. However, it will select all the formulas, not just the ones that are constant formula cells. Picking out the constant formula cells is something that would have to be done by inspecting each selected formula individually (using the Tab key to move from one cell in the selection to the next).

2.25 Search | Unreferenced cells

This command selects all cells within the current selection that are not referenced by any formulas within the workbook. These cells therefore constitute the outputs of a model.

WHY WOULD YOU WANT TO USE IT?

Best practice in the development of spreadsheet models dictates that the model should only calculate those outputs that are required and that all other calculations are intermediate steps towards the final outputs. Unreferenced cells are therefore, by definition, part of the output of a model or should be removed from the model. You can use this command to help confirm that only those cells that are intended as outputs are unreferenced. Those cells that have unintentionally been left unreferenced can be identified and fixed; any extraneous formulas can also be removed from the model.

HOW TO USE IT

Select a range of cells, or just one cell to search the entire worksheet, and choose the OAK Development/Review | Search | Unreferenced cells command. A secondary dialog box will appear, asking if you want to limit your search to cells which contain: formulas, constant numeric values, constant text values, constant logical values or some combination of these.

Selecting formulas only will cause OAK to ignore cells which contain inputs, which is often a useful feature. In order to check for unreferenced inputs, select the checkboxes that correspond to the type or types of input you wish to check, e.g. constant numeric values. Note that limiting the types of cells you wish to check will speed up the search.

If no checkboxes are selected then OAK will search for all unreferenced cells. This means that selecting every checkbox is equivalent to leaving all checkboxes unselected.

After you have clicked ok, the command will then either select those cells in the current selection that are not referenced by any formulas in the workbook or return a “No unreferenced cells found in selection” message.
Note that neither the OAK unreferenced cell command nor the alternatives given below are able to detect if a cell is referenced via Excel’s offset or indirect functions. This means that a cell referenced via the offset or indirect functions will be shown as an unreferenced cell even though it does have dependents.

WRINKLES

This operation can take a long time. The reason is that every formula in the selection must be tested for dependents.

WHAT YOU COULD DO IF YOU DIDN'T HAVE OAK

If didn't have OAK, and wished to check whether a single cell has been referenced, then Excel contains two commands that will find the dependents of that cell.

The first is the go to dependents command which is run by selecting Home | Editing | Find & Select | Goto Special and then the dependents option. This will either highlight the cells that depend on the cell that was selected or will display a “No cells were found” message if the cell is unreferenced.

The second command is the trace dependents command which is run by selecting Formulas | Formula auditing | Trace dependents. This command will draw blue arrows from the selected cell pointing to dependent cells on the same worksheet. Dependent cells on different worksheets are indicated by a broken arrow pointing to a worksheet symbol.

Double clicking on the arrow will bring up a list of dependent cells on other worksheets. If the cell is unreferenced a message stating “The Trace Dependents command found no formulas that refer to the active cell” will be displayed.

2.26 Search | Conditional search

This command selects all the cells within the current selection that match conditional criteria chosen by you.

You can choose between three types of conditional search criteria:

- numerical cell values can be compared to threshold values;
- cell values can be searched for a particular string;
- cell formulas can be searched for a string.

WHY WOULD YOU WANT TO USE IT?

There are numerous occasions when you might want to perform a search which is more complex than allowed for by Excel's built-in functions:
Using OAK: OAK commands

- you may want to search for values within a particular range, say all the cells that contain values between 90% and 100%;
- conversely, you may want to exclude a range of values and find all the cells whose values are not between 25 and 50;
- you may want to find all cells whose values exactly match a certain figure, or all cells which don’t match a particular value;
- you may want to find all cell values which are greater than or less than a certain threshold figure;
- you may want to find all the cells which contain a certain text string, say “input”;
- you may also want to search all the cell formulas for a particular text string, say “IF”, “SUM”, or “TRUE”.

HOW TO USE IT

Select a range of cells and choose the command from the menu; the Conditional Search Options dialog box is displayed. Choose one of the three following search types from the text box on the left:

- Cell Value Is;
- Cell Value Contains;
- Cell Formula Contains.

If you are performing a numerical search (‘Cell Value Is’), choose one of the following eight options:
- between;
- not between;
- equal to;
- not equal to;
- greater than;
- less than;
- greater than or equal to;
- less than or equal to.

Type in a value or text in the text box or boxes on the right. If you are performing a text search (‘Cell Value Contains’ or ‘Cell Formula Contains’), check the ‘Match case’ box if you want a case-sensitive search. Finally, click the ‘OK’ box.

The command will either select those cells within the current selection that match the selected criteria or return a “No cells in selection match conditional criteria” message.
WRINKLES

All of OAK’s Search selection commands follow the One-cell rule.

The OAK Development/Review | Search | Conditional search command resembles an amalgam of two of Excel’s built-in commands. It is like Excel’s

- Home | Editing | Find & Select | Find command, in that it looks for cell contents specified by the user
- Home | Editing | Find & Select | Goto Special command, in the way it presents the results as a selection of potentially many areas.

WHAT YOU COULD DO IF YOU DIDN'T HAVE OAK

If you did not have OAK, you would have to do any searching using Excel’s Home | Editing | Find & Select | Find command. Searching for cells in which its value or formula equals or contains a known threshold value or text string can be achieved by Excel’s Find command, shortcut: Ctrl+F, and clicking on ‘Options’ button toward the bottom right of the dialog box that appears.

Enter the threshold value or text string required in the ‘Find what’ text box and check ‘Match case’ and ‘Match entire cell contents’ as necessary. The ‘Within’ drop-down menu will allow the user to perform the search either within the worksheet or within the workbook. The ‘Look in’ drop-down menu allows the user to choose whether the search is performed on the cell formula, cell value or cell comment. Clicking on ‘Find All’ will generate a list of links to cells which match the criteria. Repeatedly clicking on ‘Find Next’ will cycle through the cells identified that match the search criteria.

Confining the search to the worksheet will make Excel perform the search subject to the One-Cell rule. Therefore, if more than one cell is initially selected and this search is performed on the worksheet, then Excel will only identify the cells within the initial selection which match the criteria. In this case, Excel has not actually performed the search on the entire worksheet which could lead to error. When performing this search on the entire workbook, Excel identifies all cells matching the search criteria within the workbook regardless of the initial selection.

The OAK conditional search command offers the advantages

- of being able to perform a “Cell Value is between...” conditional search which cannot be performed using Excel’s built-in functions
- that the cells matching the search criteria are identified and selected all at once as result of the conditional search whereas the search as performed by Excel’s built-in functions will only enable the user to select the identified cells one at a time.

2.27 Search | References to blank cell

This command selects all cells on the current worksheet that reference blank cells within...
the current selection (which is taken to include the blank cells in the unused area that surrounds each worksheet). This command is most useful when the entire worksheet (or equivalently a single cell) is selected.

WHY WOULD YOU WANT TO USE IT?

Good practice in developing models dictates that formulas should not refer to blank cells. Either the reference itself is not required and should be deleted or the cell should contain another formula or some input data. This function allows you to locate those cells that refer to blank cells and to confirm that nothing has been deleted by mistake. The formulas and input data can then be reinstated or the references deleted as appropriate.

Another danger surrounding blank cells is that it is possible that the user does not realize that a particular blank cell is referenced and introduces some unexpected data or formula to that cell. This information can “leak” into the body of the model and produce unexpected results.

HOW TO USE IT

Select a range (typically an entire worksheet or equivalently a single cell on the worksheet) and choose the command from the menu; the command will either select those cells on the current worksheet that reference blank cells within the selection (which is taken to include the blank cells outside the used range) or return a “No references to blank cells found” message.

WRINKLES

Any blank cells that are referenced on a different worksheet are not selected by this command.

Some people intentionally include blank cells in summary formulas for the sake of simplicity, for example “=SUM(D5:D125)”. To avoid finding these cells, do not include them in your selection.

WHAT YOU COULD DO IF YOU DIDN'T HAVE OAK

If you didn't have OAK, you could find find blank cells that are being referenced by other cells using Excel’s Go to | Special command. Selecting the entire worksheet, then using Home | Find & Select | Goto Special | Precedents will select all the cells that are being referenced by other cells. Using Home | Find & Select | Goto Special | Blanks now will then return all blank cells that are being referenced, which can then be checked and edited as needed.

What takes Excel two operations is done more neatly by OAK in a single action.
2.28 Summarize

In the business world, the individual who builds a particular spreadsheet is rarely the only person to use it.

As either a spreadsheet builder or a spreadsheet user, it is often convenient to have the capability of producing a summary of a particular workbook. For those who build spreadsheets in the first place, such a facility would help to detect errors before releasing models. For end-users, it provides an overview of the logic of the model.

It would also be extremely valuable if you could state in a few words how complex a given workbook is. This information can help you to estimate how long it might take for the average user to learn to use the workbook effectively.

Of course, Excel workbooks can vary enormously in both size and complexity. Depending on the spacing between formulas, a workbook which contains many worksheets can be simpler than a workbook containing a single worksheet. Thus it is difficult to judge the complexity of a given workbook from its size alone.

Furthermore, even if you know the number of formula cells in a given workbook, you would still need some measure how many different, or distinct, formulas are present. Even nicer would be a measure of the complexity of the individual distinct formulas; this would allow a reasonable estimate of how difficult the workbook might be to understand.

The OAK Review | Summarize command provides you with just this kind of information about an Excel workbook.

OAK WORKBOOK / WORKSHEET SUMMARY

The OAK Review | Workbook | Summarize command allows you to establish key facts about the size and complexity of a workbook.

This command can generate four reports:

- a workbook summary that lists all the selected worksheets contained within the workbook, with detailed statistics for all the unprotected worksheets;
- a list of all distinct formulas (on unprotected worksheets) in the selected worksheets, along with details for each, including a measure of their complexity;
- a list of all referred constants in the selected worksheets, along with details for each.
- a list of all distinct formulas (on unprotected worksheets) in the selected worksheets, along with a count of the occurrences of a selection of formula elements, and a set of configurable risk coefficients associated with each of these elements, which together produce an overall risk score for the selected worksheets.

Note that if you want OAK to provide statistics for the protected worksheets in a workbook, you must first unprotect them.
WHY WOULD YOU WANT TO USE IT?

The Summarize command is useful in the following situations:

- when you first encounter a model, the workbook summary gives a measure of how large and complex it is and allows you to estimate how long it will take to understand it;

- similarly, when you first encounter a model, the formula list helps you to understand the complexity of individual formulas and to identify those that may require particular attention (i.e. those that are the longest);

- when you are producing model documentation, you may require an overview of the model structure and a list of all the formulas contained in the model;

- when you are ready to release a model, you will need to confirm that it contains no unknown error cells.

- when you are reconciling model inputs with model documentation.

HOW TO USE IT

When you choose the Summarize command from the OAK menu, a dialog box is displayed, giving you a few options to choose from.

The Summarize Workbook, List Distinct formulas and List Referred Constants generate different reports. An additional option, Output in Same Workbook, simply defines where to place the reports. The lists allow you to choose which workbook, or which individual sheets within that workbook to summarize.

WORKBOOK SUMMARY

The workbook summary report lists the sheets selected to be summarized and analyzes the unprotected worksheets for the following information:

- numbers of non-blank rows and columns, which are multiplied together to give the effective number of cells in use (this is not the number of non-blank cells);

- an indication of whether the used range (located using Ctrl+End or the Home | Editing | Find & Select | Go To Special | Last Cell command, see Chapter 4) is significantly outside the range of non-blank cells;

- number and percentage of the cells in use containing formulas;

- number and percentage of the formula cells that contain distinct formulas; a distinct formula is one which is inconsistent with the formula in the cell to its left and with the formula in the cell above (see Chapter 4 for details);

- number and percentage of the formula cells which contain error values;

- number and percentage of the cells in use containing constants;
• number of individual arrays, number of array cells, and the percentage of the cells in use which are array cells;

• number of merged cells and the percentage of cells in use which are merged

• Last Cell Location. This column on the workbook summary report identifies worksheets whose used range, as defined by Excel, is much larger than the region on the worksheet which actually contains information. The discussion on used range explains how this can happen, why it's inconvenient, and what you can do about it.

• Formula Cells, Distinct Formulas, Constant Cells, Array Cells. These columns provide you with a rough idea of the overall logic and complexity of the model. A high proportion of constants on a worksheet indicate that the sheet is an input sheet. By contrast, a high proportion of formulas indicate that the worksheet is dominated by calculations. Finally, a high proportion of distinct formulas and arrays indicate that the worksheet is complicated.

• Error Cells. This column indicates the presence of Excel error cells and can be used to confirm that a workbook does not contain any error cells when it is released. These cells are either constant or formula cells which result in one of the following: #NULL!, #DIV/0!, #VALUE!, #REF!, #NAME?, #NUM!, or #N/A.

• Merged Cells. This column tells you if there are any merged cells on the worksheet. Operis finds the presence of merged cells in a spreadsheet to be inconvenient. This column gives an indication of how affected by this issue are the worksheets being studied.

NOTATION

If the used range includes too many blank rows (at the bottom) or too many blank columns (on the right), this is indicated using the following text strings:

• “>Rows”, which means there are between 10 and 20 blank rows at the bottom of the used range;

• “>>Rows”, which means there are more than 20 blank rows at the bottom of the used range;

• “>Columns”, which means there are between 10 and 20 blank columns on the right-hand edge of the used range;

• “>>Columns”, which means there are more than 20 blank columns on the right-hand edge of the used range.

DISTINCT FORMULA LIST

The distinct formulas report lists all of the distinct formulas in the workbook or selected worksheets. A distinct formula is one which is inconsistent with the formula in the cell to its left and with the formula in the cell above (see Chapter 4 for details). The report provides the following information for each formula:
Using OAK: OAK commands

- the worksheet which contains the formula;
- the address of the first cell which contains the formula;
- the complexity of the formula;
- the number of cells which contain the formula;
- a flag to indicate whether the formula refers to a range on a different worksheet (“Off Sheet”) or workbook (“Book Link”); see the “External Refs” section below;
- the formula itself.

The most interesting columns are those with the headings Complexity, Frequency, and External Refs.

- Complexity. This column shows the number of elements which make up the formula, and is therefore useful in estimating how complicated each formula is. Sorting the list in descending order by this column identifies the most difficult formulas in the workbook, and the ones which are the most likely to contain errors.
- Frequency. This column simply tells you how many cells contain the same formula. If your worksheets follow good practice in developing models, this column will tend to contain two values: 1 and N, where N is the width of your model.
- External Refs. This column tells you if the formula refers to a cell on another worksheet or workbook. The flag “Off Sheet” is used to indicate that the formula refers to a range on another worksheet using cell references or local names, but not if it does so using global names. The flag “Book Link” is used to indicate that the formula refers to a range in a different workbook.

REFERRED CONSTANTS LIST

The referred constants report lists all of the referred constants in the workbook or selected worksheets. A referred constant is a cell that contains an alpha/numeric value that is referred to by at least one other cell. The report provides the following information for each constant cell:

- the worksheet which contains the formula;
- the address of the cell which contains the constant;
- the value of the constant.

This allows the constants used in the model to be checked against related documentation.

RISK ANALYSIS

The risk analysis report lists all of the distinct formulas in the workbook or selected worksheets, along with a count of the number of occurrences of a selection of formula elements. These are summed, and the total for each element is multiplied by a corresponding risk weighting coefficient, and these products are summed to produce an overall risk score for the workbook or selected worksheets.
These analyzed elements are:

<table>
<thead>
<tr>
<th>Element</th>
<th>Default Risk Weighting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nested 3+</td>
<td>1</td>
<td>Elements nested 3 or more levels deep</td>
</tr>
<tr>
<td>Nested 5+</td>
<td>5</td>
<td>Elements nested 5 or more levels deep</td>
</tr>
<tr>
<td>Nested 7+</td>
<td>10</td>
<td>Elements nested 7 or more levels deep</td>
</tr>
<tr>
<td>OFFSET</td>
<td>25</td>
<td>The OFFSET function</td>
</tr>
<tr>
<td>INDIRECT</td>
<td>25</td>
<td>The INDIRECT function</td>
</tr>
<tr>
<td>LOOKUPs</td>
<td>5</td>
<td>The LOOKUP, VLOOKUP and HLOOKUP functions</td>
</tr>
<tr>
<td>IFERROR</td>
<td>3</td>
<td>The IFERROR function</td>
</tr>
<tr>
<td>Hardcoded Constants</td>
<td>2</td>
<td>Values hardcoded into the formula. 0 and 1 are excluded.</td>
</tr>
<tr>
<td>Matched Names</td>
<td>-2</td>
<td>Names defined on a range of potentially more than 1 cell, that contribute only one cell value to the formula, by way of column or row matching</td>
</tr>
<tr>
<td>Unmatched Names</td>
<td>-2</td>
<td>Names that are not column or row matched</td>
</tr>
<tr>
<td>Cell References</td>
<td>0</td>
<td>Cell or range references</td>
</tr>
<tr>
<td>Cells Referenced</td>
<td>0</td>
<td>An attempt to calculate the number of cells referenced by the formula. This is not always possible, with variable ranges and functions like OFFSET and INDIRECT.</td>
</tr>
<tr>
<td>Variable Ranges</td>
<td>0</td>
<td>Variable ranges, for example, an absolute to relative range: e.g. the R1C1 formula R1C1:R[-5]C[-5], which covers a different area depending on the location of the formula.</td>
</tr>
<tr>
<td>Errors</td>
<td>2</td>
<td>Error values</td>
</tr>
</tbody>
</table>

The risk weightings in the report, which are located in the row titled "Risk Weighting" on the left of the report, can be modified, and the report will recalculate the totals based on the new risk weightings.

These characteristics, why there are risky, and what can be done to improve them, are discussed in Chapter C4.

That discussion necessarily reflects Operis views about what is good or poor spreadsheet programming practice. Others will hold different opinions. The OAK risk analysis can accommodate these diverging approaches. It gives weights reflecting the severity of each questionable practice which are combined to give an overall risk score for a worksheet. The default weights can be replaced by others of the user’s choosing to give his own version of a risk score.

WRINKLES

Summarize workbook command cannot analyze worksheets which are protected. Neither the Workbook Summary nor Distinct Formulas reports will provide information about protected worksheets.
Operis notes that there are several different approaches to distinct formulas in cases where formulas are identical but located in different parts of a worksheet or in different worksheets. It may be the case in other packages that an aggressive approach is used, where if a formula is identical to another but separated by a blank line, they still count as one distinct formula. An even more aggressive approach may also be used, where provided a formula is identical to another formula in a different part of the worksheet they are counted as only one distinct formula, no matter where on a worksheet that formula may be located. OAK, however, uses a prudent approach, where as long as the formula is dissimilar to the formula in the cell above it or to the left of it, it is counted as a distinct formula.

Due to the limitations of Excel, this report cannot analyze very long formulas. Instead, the following message is displayed:

"***** WARNING: FORMULA TOO COMPLEX TO RETURN *****". Formulas reported in this way should be modified if at all possible.

WHAT YOU COULD DO IF YOU DIDN'T HAVE OAK

OAK is not the only package to offer a function equivalent as Summarize Workbook. Many other Excel packages will be able to perform a similar task.

Things that are distinctive about OAK's Summarize Workbook are

- that the reports are prepared exceptionally fast: generally a few seconds. Some packages, including versions of OAK before 3.5, can take over an hour to summarize a large spreadsheet.
- the cautious approach that OAK takes in deciding which formulas are distinct

It is also possible in Excel to be able to review the formulas in each cell by using Excel's Formulas | Formula Auditing | Show Formulas command. The shortcut for this command is Ctrl+`. (On a British keyboard, this is an obscure key between the Tab and Escape keys, to the left of the number 1.)

2.29 Columns and Rows | Insert

Inserting rows or columns into a spreadsheet is such a basic operation that Excel offers a really quick short cut for it.

- Select some rows or columns (Shift+Space and Ctrl+Space are useful here)
- Press Ctrl+Plus (the + key on the numeric pad is the most convenient one to use here)

Just one problem: the command doesn't work if any of the rows or columns to be inserted intersect any cells that

- have been merged
- form part of an array constant or array formula
• form part of a data table.

OAK’s Insert Rows/ Columns commands do allow you to insert rows and columns through arrays and merged cells. In addition, the Insert Rows/ Columns command has an option which copies formulas into the newly inserted region; this option will only be of interest to users who follow good practice in developing models and use left-to-right, or top-to-bottom, consistent formulas.

WHY WOULD YOU WANT TO USE IT?

Spreadsheet users often find they need to extend the width of a model. Perhaps the project lifetime has been extended or the initial estimated horizon was too short. When you encounter such a problem, you typically have two choices: to manually extend the model into the new columns or rows, or to insert new columns or rows through the middle of the model.

The latter approach means that you must insert a row or a column through existing formulas, but this has the enormous benefit of extending any named ranges through which the insertion was made. The former approach, by extending formulas at the end of the model, does not do this, which means that many named ranges must be recreated.

Therefore, insertion appears to be the way forward. However, there are two drawbacks to using the standard Excel insert commands. One is that Excel will not allow the insertion to take place if the row or column intersects an array, data table, or a merged cell. The other is that you must copy the formulas and values into the inserted region.

The OAK Insert Rows/ Columns command removes both of these obstacles. You can insert a row or a column through an array, data table (though not a pivot table), or merged cell, and you are offered the option to copy all the formulas across into the newly inserted region.

HOW TO USE IT

The OAK insertion command is used in a slightly different way from the Excel insert function. You must first select the entire row or column you would like to insert, and then use the appropriate OAK command. You can select more than one row or column to insert or delete at a time. There are useful shortcut keys for selecting a column (Ctrl+Space) or row (Shift+Space).

When you use the OAK Development | Worksheet | Columns and Rows | Insert command, a secondary dialog is displayed. This dialog asks whether you want formulas to be copied from the left or from above into the newly created row or column. This is a very useful option if your formulas are left-to-right or top-to-bottom consistent.

If you click the ‘Yes’ button, the row or column is inserted and the formulas are copied across consistent with the adjacent formulas. Note, however, that constants are not copied across – you must enter them explicitly.
If you click the ‘No’ button, the row or column is inserted, but is left blank.

Clicking the ‘Cancel’ button causes the dialog to disappear without doing anything.

The Insert Rows/Columns command works exactly the same way when inserting multiple columns or rows. Select the number of rows or columns to be inserted and choose the command from the menu. Again, you have the option of copying the formulas across.

The option which allows you to copy formulas into the newly inserted rows or columns is only useful when the formulas are consistent from cell to cell in the same row or column. This option has been included only for convenience and users of this OAK function are advised to double-check that any formulas pasted as a result of this are as required. The following points should be noted when using the Insert and Delete Rows/Columns commands.

**HOW IT WORKS**

OAK’s insert rows/columns command functions by detecting merged cells, arrays and data tables; turning them into unmerged cells with simple scalar formulas in them; having Excel perform the insertion in the usual way; and then restoring all the merger cells, arrays and data tables.

**WHAT YOU COULD DO IF YOU DIDN’T HAVE OAK**

There is no obstacle in principle to doing the things that OAK does by hand through the Excel user interface. But it would be tiresome and difficult to do accurately.

**WRINKLES**

When inserting a row or column within an array, it will be likely that because a similar change may not have been implemented on the “area” of cells to which the array formula refers, “#N/A” errors may result within the array.

Insertion of rows or columns within an array will mean that any previously left-to-right or top-to-bottom consistent (scalar) formulas that refer to the cells (by use of cell reference instead of names) within array will now not be consistent and would require amendment by the user. In the case of deleting rows or columns from an array, any left-to-right or top-to-bottom consistent (scalar) formulas which refer to cells in the array would show “#REF!” errors.

There are a couple of slight difference between the way in which the Excel and the OAK insertion and deletion commands work.

- If multiple worksheets are selected, the Excel commands apply to all of them, while the OAK commands apply only to the active worksheet.
- Excel offers the option to Undo its insertions but OAK does not offer an Undo facility for this or any other command.
2.30 Columns and Rows | Delete

Deleting rows or columns from a spreadsheet is such a basic operation that Excel offers a really quick short cut for it.

- Select some rows or columns (Shift+Space and Ctrl+Space are useful here)
- Press Ctrl+Minus (the minus key on the numeric keypad is the most convenient one to use here)

Just one problem: the command doesn't work if any of the rows or columns to be inserted intersect any cells that

- have been merged
- form part of an array constant or array formula
- form part of a data table.

OAK’s Delete Rows/Columns commands do allow you to delete rows and columns even if they pass through arrays and merged cells.

WHY WOULD YOU WANT TO USE IT?

Many financial models are based on templates, intended as the starting point for models of a variety of deals. Sometimes the templates contemplate the possibility of a long project but you are modelling a short project. Chopping out the unnecessary columns will make the model smaller, quicker to load, quicker to calculate, and easier to email.

However, the standard cell delete operation built into Excel will not allow the deletion to take place if the row or column intersects an array, data table, or a merged cell.

The OAK Delete Rows/Columns command overcomes this obstacle. You can delete a row or a column that passes through an array, data table (though not a pivot table), or merged cell.

HOW TO USE IT

The OAK deletion command is used in a slightly different way from the Excel delete function. You must first select the entire row or column you would like to delete, and then use the OAK Development | Worksheet | Columns and Rows | Delete. You can select more than one row or column to delete at a time.

HOW IT WORKS

OAK’s delete rows/columns command functions by detecting merged cells, arrays and data tables; turning them into unmerged cells with simple scalar formulas in them; having Excel perform the deletion in the usual way; and then restoring all the merged cells, arrays and data tables.
WHAT YOU COULD DO IF YOU DIDN'T HAVE OAK

There is no obstacle in principle to doing the things that OAK does by hand through the Excel user interface. But it would be tiresome and difficult to do accurately.

WRINKLES

When inserting a row or column within an array, it will be likely that because a similar change may not have been implemented on the “area” of cells to which the array formula refers, “#N/A” errors may result within the array.

Deletion of rows or columns within an array will mean that any previously left-to-right or top-to-bottom consistent (scalar) formulas that refer to the cells (by use of cell reference instead of names) within array will now not be consistent and would require amendment by the user. In the case of deleting rows or columns from an array, any left-to-right or top-to-bottom consistent (scalar) formulas which refer to cells in the array would show “#REF!” errors.

There are a couple of slight difference between the way in which the Excel and the OAK deletion commands work.

- If multiple worksheets are selected, the Excel commands apply to all of them, while the OAK commands apply only to the active worksheet.
- Excel offers the option to Undo its deletions but OAK does not offer an Undo facility for this or any other command.

2.31 Transpose

OAK’s Transpose command copies cells from the one area of the active worksheet to another, rotating their orientation, so that what previously was arranged along rows becomes listed down columns.

WHY WOULD YOU WANT TO USE IT?

As noted repeatedly in this help, Operis believes strongly that the best way to test a spreadsheet is to prepare reconstructions of the key calculations, in a way that is intentionally different from the original.

One way to build up a parallel calculation that is different from the original is to make a different choice of the dimensions along which to lay it out. In the illustration below, a model under test that has time and revenues/costs as primary categories, with territories as a subcategory, is reconstructed using time and territories as the primary categories, with revenues/costs as the subcategory.
As can be seen from the shaded areas, which indicate select cells that are equivalent in the two analyses, the numbers in the reconstruction are at right angles to the original values. Any linkage between the two can be established by

- selecting the the original cells in the model under test
- copying them to the clipboard
- switching to the window containing the reconstruction spreadsheet
- using Excel’s Paste Link command
- while the newly linked cells are still selected, invoking OAK’s Transpose command to flip them through 90 degrees.

**HOW TO USE IT**

To transpose the cells in place
1. Select the cells you want to transpose
2. Invoke the OAK4 | Range | Transpose command
3. The selected cells will be transposed.

To make a transposed copy of the cells at another place
1. Select the cells you want to copy.
2. Press and hold Ctrl and click on a single cell which will be the top-left corner of the area copied to. (This selects a multi area range of 2 areas).
3. Invoke the OAK4 | Range | Transpose command.

### Model under test

<table>
<thead>
<tr>
<th>Year</th>
<th>2011</th>
<th>2012</th>
<th>2013</th>
<th>2014</th>
<th>2015</th>
</tr>
</thead>
<tbody>
<tr>
<td>Revenue</td>
<td>Americas</td>
<td>Europe</td>
<td>Africa and Middle East</td>
<td>Asia and Australasia</td>
<td></td>
</tr>
<tr>
<td>Variable costs</td>
<td>Americas</td>
<td>Europe</td>
<td>Africa and Middle East</td>
<td>Asia and Australasia</td>
<td></td>
</tr>
<tr>
<td>Fixed costs</td>
<td>Americas</td>
<td>Europe</td>
<td>Africa and Middle East</td>
<td>Asia and Australasia</td>
<td></td>
</tr>
</tbody>
</table>

### Reconstruction

<table>
<thead>
<tr>
<th>Area</th>
<th>Americas</th>
<th>Europe</th>
<th>Africa and Middle East</th>
<th>Asia and Australasia</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figures for 2011</td>
<td>Revenue</td>
<td>Variable costs</td>
<td>Fixed costs</td>
<td></td>
</tr>
<tr>
<td>Figures for 2012</td>
<td>Revenue</td>
<td>Variable costs</td>
<td>Fixed costs</td>
<td></td>
</tr>
<tr>
<td>Figures for 2013</td>
<td>Revenue</td>
<td>Variable costs</td>
<td>Fixed costs</td>
<td></td>
</tr>
<tr>
<td>Figures for 2014</td>
<td>Revenue</td>
<td>Variable costs</td>
<td>Fixed costs</td>
<td></td>
</tr>
<tr>
<td>Figures for 2015</td>
<td>Revenue</td>
<td>Variable costs</td>
<td>Fixed costs</td>
<td></td>
</tr>
</tbody>
</table>
4. The contents of the source cells will be copied to a transposed, but otherwise equivalent-sized, rectangle anchored on the indicated position.

WRINKLES

This command does not work when applied to protected worksheets. You must manually unprotect them first.

The destination cell should be a single cell. It must be blank, since otherwise it is not clear which cell is to be copied. If no destination cell is specified, it is taken to be the top left corner of the selection.

If the formula being transposed acts on a range of several cells and identifies them by user-defined names, it is necessary to replace the names by coordinate references before copying them, since the row/column matching on which such names typically rely is most unlikely to work when the formulas have been transposed. OAK will perform this action. In fact, it will remove the names even when it isn't strictly necessary, for example when a single cell is identified by name.

WHAT YOU COULD DO IF YOU DIDN'T HAVE OAK

Excel offers its own means to transpose formulas:

- Highlight a range and copy it onto the clipboard.
- Select a destination cell.
- Use Excel's Home | Clipboard | Paste | Transpose option.

But there are several differences between the Excel command and the OAK one which in many cases make the difference between an action that is useful in several circumstances and one that is useful only rarely.

- Excel adjusts the formulas so that any relative column references become equivalent relative row references, and vice versa. So, using R1C1 notation, =RC[1] becomes =R[1]C after transposition. The result is that the formulas refer to different cells after they have been transposed, unless those referenced cells have themselves been transposed at the same time. OAK makes no such adjustment, so that although the formulas are transposed, they still refer to the cells that they mentioned before, which is what is wanted if they are devoted to plucking numbers out of another worksheet in order to validate them. (Several tricks for defeating Excel's alteration of these formulas are offered in the discussion of OAK's Copy Literal command)

- The Excel command makes no adjustment to a formula's uses of Excel names. Since such formulas often rely on Excel's column or row matching, they will cease to work after transposition. As already noted, OAK addresses this limitation by stripping out any names, substituting coordinate notation, before doing any transposition.

- An attempt to use Excel's transpose command to transpose cells in place will result in the complaint that "The selection is not valid". Cells have to be transposed to a location
that does not overlap the original. OAK stores the cell contents before transposing them, so it is able to overwrite the original cells with the new contents.

- If you copy a small number of cells and paste them into an area consisting of a larger number of cells, Excel’s Paste command will repeat the source material as necessary to fill the target area. No equivalent action is possible with OAK’s Transpose, as it doesn’t offer the opportunity to specify the target area with more than one cell.

2.32 Copy Address

OAK’s Copy Address command copies the address of the current selection to the clipboard, using a variety of formatting options.

WHY WOULD YOU WANT TO USE IT?

Someone checking a spreadsheet may wish to include in a written report the location of issues that have come to light in the course of a review. Where the cells involved are numerous, and widely scattered over a worksheet, writing out the location can be tedious and hard to achieve with accuracy, particularly if there is quite a number of issues to report.

HOW TO USE IT

1. Select the cell(s) for which you want to copy the addresses.
2. Go to OAK Development | Cells | Copy Address
3. The Copy Address dialog appears. Use this to select the formatting options. Click OK to place the text in the clipboard.
4. Paste the text into the target document.
FORMATTING OPTIONS

- Prefixing
  - Include Workbook Name: Includes the workbook name in the prefix. Cannot be used without inclusion of the worksheet name.
  - Include Worksheet Name: Includes the worksheet name in the prefix.
  - Prefix all regions: Prefix all the regions listed, rather than just the first. Prefixes on every listed region can appear superfluous in natural language documents.
  - Notation: Select between A1, $A$1 and RC notation.
  - Delimiter: Select the character used to delimit the listed regions, optionally with an additional space.

WHAT YOU COULD DO IF YOU DIDN'T HAVE OAK

One can achieve an approximation to the Copy Address operation without using OAK, by invoking Excel's Insert | Name | Define command to bring up the Define Name dialog. The bottom field of that will contain a text representation of the selected cells. Press Tab twice to move the cursor into this field, and the text will become highlighted. Now press Ctrl-C to copy it onto the clipboard, and Esc to dismiss the dialog.

2.33 Copy Literal

OAK's Copy Literal command copies cells from the one area of the active worksheet to another, without adjusting cell coordinates as Excel normally does.

When a cell containing a formula is copied to another location using the copy and paste commands, Excel goes to great trouble to adjust the cell references in the formula to reflect its new position.

For example if the cell A1 contains the formula =B3 and this cell is copied to B2, then the formula in cell B2 will be =C4, i.e. since the cell containing the formula has been copied one column across and one row down, the cell reference has been moved one column across and one row down. This behavior is generally exactly what is needed and is so automatic for spreadsheet users that they rarely think about the behaviour.

However there are occasions when the adjustment is very much not wanted, and the wish is to make to copy the exact contents of a cell, so that in the example above the cell B2 would contain the formula =B3. This is what the Copy Literal function does.

Other ways to express the action of OAK's Copy literal include
Rather than copying cells, it copies the contents of those cells

When Excel copies formulas, it preserves them in R1C1 notation, which has the effect of causing them to change when shown in A1 notation. OAK's Copy literal command preserves them in A1 notation, which has the effect of causing them to change when shown in R1C1 notation.

It is like cutting cells and pasting them elsewhere, but without deleting the originals.

WHY WOULD YOU WANT TO USE IT?

When checking a spreadsheet, Operis typically develops many reconciliations and tests. To avoid altering the original spreadsheet, it places the reconciliations and tests on a second spreadsheet. Operis refers to the first spreadsheet as the MUT (Model Under Test) and the second as the AWP (Audit Working Papers).

The second spreadsheet is full of formulas of the kind =WorkbookUnderTest[Sheet1]!A1, so that it can extract values from the first spreadsheet for examination. Such link formulas can be introduced:

- either, by hand: select a cell in the second spreadsheet, type an =, switch windows to the first spreadsheet, navigate to the top left corner of the region to be linked, press Enter, copy the formula as necessary
- or, automatically: select the region in the first spreadsheet, copy it on to the clipboard, switch windows to the second spreadsheet, navigate to a suitable place in it, use Home | Clipboard | Paste | Paste Link.

The second method is fractionally quicker and is often used by experienced analysts.

The spreadsheet being tested will typically have some hotspots that are the subject of several tests. If it generates financial statements, they are likely to be one such example. It may therefore be desirable to copy the link formulas from one test to begin another test.

Had the link formulas been hand-typed using the first method, the author might have anticipated the possibility of having to make further copies of them elsewhere in his worksheet, and added appropriate dollar signs to control the process. But the formulas inserted by Excel using the second, quicker method have relative references, unless there is just one cell involved. Any copy made by Excel's copy and paste (or equivalent methods) will point not at the hotspot originally selected, but at some cells that are offset from there.

HOW TO USE IT

1. Select the cell(s) you want to copy.
2. Go to OAK Development | Cells | Copy Literal
3. A range selector dialog appears. Use this to select the destination. Click OK.
4. The exact contents of the source cell will be copied to the destination cell, prefixed accordingly if the destination is on a different worksheet.

OAK offers an alternative selection mode for the Copy Literal function, called Double-select mode. This mode can be enabled on the User Interface tab in the Options window. Once this has been enabled, do the following:

1. Select the cell(s) you want to copy then press and hold Ctrl and click on the cell you want to copy to.
2. Go to OAK Development | Cells | Copy Literal
3. The exact contents of the source cell will be copied to the destination cell.

WRINKLES

This command does not work when applied to protected worksheets. You must manually unprotect them first.

The destination cell should be a single cell. In double-select mode it must be blank, since otherwise it is not clear which cell is to be copied.

If the formula being copied acts on a range of several cells and identifies them by user-defined names, it is necessary to replace the names by coordinate references before copying them. OAK will perform this action. In fact, it will remove the names even when it isn't strictly necessary, for example when a single cell is identified by name.

If you copy a small number of cells and paste them into an area consisting of a larger number of cells, Excel's Paste command will repeat the source material as necessary to fill the target area. OAK's Copy Literal won't do this. It will take the target cell as its anchor, and place a single copy of the source cells there.

WHAT YOU COULD DO IF YOU DIDN'T HAVE OAK

Competent Excel users will be able to think of several ways to circumvent the adjustment to coordinates that Excel performs when it copies formulas.

- Adjust the cells by making cell references absolute; ie insert dollar signs as necessary.
- Copy the literal contents of a cell. First select the cell you wish to copy, then press F2; this allows you to edit the contents of the cell. Now press Shift+Home to highlight the contents of the cell and press Ctrl+C to copy them. Then select the destination cell and press Ctrl+V. The literal contents of the source cell have now been copied to the destination cell. (But it's painful to do this for a range consisting of many cells.)
- Highlight a range. Use Excel's Replace command to change the Equals signs to something else. (Before OAK had Copy Literal, Operis typically used &&& or ^^^, since they are unlikely to appear in a formula naturally) Since Excel expects formulas to start with equals signs, it will no longer consider that the cells contain formulas, and will copy
them without coordinate adjustment. Excel's Replace command can then be used to restore the equals signs.

And there are more. But OAK’s Copy Literal command is quicker than any of them.

### 2.34 Remove Color Formatting

The OAK Development | Worksheet | Remove Color Formatting command removes all non-conditional color formatting from either entire workbooks or a selection of worksheets within a workbook.

**WHY WOULD YOU WANT TO USE IT?**

**Polite version:** Sometimes when you first receive a new model that has been built by someone else you will find that many colors are used. These colors may have been helpful to the original modeller in reminding them of what each part of the model does, but can be confusing when you try and read the model. The Remove Color Formatting tool allows you to remove these confusing colors.

**Impolite version:** In Operis's experience, the quality of a financial model is inversely related to its use of color. When Operis receives a model for review, and opens it for the first time, an initial verdict of "Very colorful" would not be a flattering one.

**Paranoid version:** Some models seek to hide calculations that don't bear close scrutiny by formatting them as white text on a white background, so that they aren't obvious to casual readers.

OAK's Remove Color Formatting command removes color formatting from the text in the cell as well as the cell background. This allows you to spot when things have been hidden by making the text the same color as the cell background.

**HOW TO USE IT**

1. Go to OAK Development | Worksheet | Remove Color Formatting

2. To remove colors from whole workbooks choose the Whole Workbook(s) option and the required workbook(s) from the list below.
   
   To remove colors from a selection of sheets within a workbook, choose the Selected Worksheets option. Then choose the workbook that contains the required worksheets from the list on the left and choose the required worksheets from the list on the right.

3. After clicking OK the colors will be removed from the required workbooks or worksheets. Cell borders will not be affected.
WRINKLES

This tool does not remove any conditional color formatting. To remove conditional formatting from a cell or selection of cells go to Home | Styles | Conditional Formatting | Clear rules from selected cells.

Excel allows a limited number of cell formats, and in versions of Excel prior to Excel 2007 this limit is sufficiently low for some developers of very colorful workbooks to encounter it. Remove Color Formatting uses a simple method of removing the formatting, but in doing so briefly increases the cell format count, so it is possible that Remove Color Formatting will not work on workbooks that have already approached the limit.

WHAT YOU COULD DO IF YOU DIDN'T HAVE OAK

The colors can be removed from a single sheet by pressing Ctrl+A to select the whole sheet, selecting No Fill from the Fill Color tool and then selecting Automatic from the Font Color tool. This method means that colors must be removed from worksheets one sheet at a time. With OAK’s Remove Color Formatting tool the colors can be removed from many worksheets at once.

2.35 Unhide rows and columns

The OAK Review | Workbook | Unhide Cells tool unhides all hidden rows and columns in either whole workbooks or a selection of worksheets from a workbook.

WHY WOULD YOU WANT TO USE IT

When building a model people sometimes hide rows and columns so that they can concentrate on the pieces of the model they are currently working on without other pieces of the model getting in the way. If a template model has been used as the basis of the new model that has been built then unnecessary parts of the template model may have been hidden.

The Unhide Rows And Columns command allows you to quickly unhide rows and columns so that you can see if there are any errors in the hidden sections. In the case where a template model has been used you can check that the hidden sections are really not used in the new model.

HOW TO USE IT

1. Go to OAK Review/Development | Workbook/Worksheet | Unhide Cells

2. To unhide rows and columns in whole workbooks choose the Whole Workbook(s) option and the required workbooks from the list below.

   To unhide rows and columns in a selection of sheets within a workbook choose the Selected Worksheets option. Then choose the workbook that contains the required
worksheets from the list on the left and choose the required worksheets from the list on the right.

3  After clicking OK all rows and columns will be unhidden on the selected worksheets.

WHAT YOU COULD DO IF YOU DIDN'T HAVE OAK

All rows and columns on a single worksheet can be unhidden by first selecting the whole worksheet by pressing Ctrl+A. Then go to Home | Cells | Format | Hide & Unhide | Unhide Rows and then Home | Cells | Format | Hide & Unhide | Unhide Columns. This means that hidden rows and columns must be unhidden one worksheet at a time. With OAK’s Unhide Rows And Columns tool rows and columns can be unhidden on many worksheets at once.

2.36 Operis Financial Modelling Book

This command in OAK presents an unashamed plug for a book, Practical Financial Modelling, which is written by Jonathan Swan, Operis’s senior trainer in financial modelling.

WHAT YOU COULD DO IF YOU DIDN'T HAVE OAK


2.37 Worksheet Manager

OAK’s Worksheet Manager is a tool for manipulating what can be a large collection of worksheets and charts in a workbook. The Worksheet Manager also deals with macro sheets and dialog sheets.

PROTECTION, CALCULATION AND VISIBILITY

The Worksheet Manager can be used to change the protection, calculation or visibility of a whole selection of sheets in one operation.

RENAMEING

The Worksheet Manager can be used to perform find and replace operations, including the use of wildcards.

SELECTION AND ORDERING

The Worksheet Manager can be used to select groups of sheets based on criteria such as:

- Ordinal position in the workbook
- Sheet type (worksheet chart, macro sheet, dialog)
Using OAK: OAK commands

- Name (with wildcards)
- Protection, calculation and visibility status

The selected sheets can then be modified or reordered as a group.

2.37.1 Managing Worksheets

OVERVIEW

The Worksheet Manager displays a grid showing a selection of properties of the sheets in the workbook. A user can manipulate these properties using the user interface. If changes have been made, the Worksheet Manager shows how the workbook will be after the changes are applied.

The columns displayed in the grid are as follows:

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>The ordinal position of the sheet in the workbook. Note that though a sheet can be moved up and down in the display order, its index will remain the same. The &quot;Set Order&quot; button is used to reindex the sheets to the displayed order.</td>
</tr>
<tr>
<td>Type</td>
<td>Indicates the Excel sheet type: Worksheet, Chart, Dialog, Excel 4 Macro, Excel 4 International Macro</td>
</tr>
</tbody>
</table>
Sheet Type (optional)  This is an optional field that displays the value of a specially selected local name defined to indicate the purpose of the sheet. It is toggled using the "Display Name" button. See section Type Name for Sheets for instructions on how to use this feature.

Sheet  The name of the sheet.

Protected  Indicates if the sheet is protected.

Calculate  Indicates if calculation is enabled for the sheet. Altering this property can have significant consequences, so there are a variety of options in the Worksheet Manager options for restricting the use of this facility.

Visibility  Shows the visibility of the sheet.

SELECTION

The selection panel is toggled by clicking the "Select" button. The following types of selection can be made:

Index  Comma-separated lists of numbers or ranges.

In a workbook with 12 sheets, the following example selects the sheets at index 1, 2, 5, 6, 7, 8, 10, 11 and 12:

1, 2, 5-8, 10-

Type  Worksheet, Chart, Dialog, Excel 4 Macro, Excel 4 International Macro

Sheet Type (optional)  One of the values selected from a list of values taken from the sheet type name as defined in the worksheets.

Sheet  A wildcard string. Use a question mark (?) to indicate a single character, and an asterisk (*) to indicate any number of any character.

Protected  Checked, unchecked or indeterminate (filled in) to ignore this property.

Calculate  Checked, unchecked or indeterminate (filled in) to ignore this property.

Visibility  Visible, Hidden, Very Hidden or blank to ignore this property.
Using OAK: OAK commands

SORTING

The displayed order of the sheets can be sorted by up to two of index, Excel sheet type, or name, in ascending (+) or descending (-) order.

COPYING

Select the sheets that are to be copied, and click the Copy button on the Worksheet Manager's toolbar. Copies of each selected sheet will be inserted immediately after their source.

The sheets are not actually copied until after the "Apply" or "OK" button has been clicked.

INSERTING

New sheets are inserted either after (default) or before the first item in the current selection.

A new worksheet can be inserted by clicking the "Insert" button.

Alternatively, the arrow beside the "Insert" button can be clicked, revealing a variety of options including other types of sheet to insert, or the option to change where the sheets are inserted, i.e. before or after the first item in the selection.

The sheets are not actually inserted until after the "Apply" or "OK" button has been clicked.

DELETING
Select the sheets that are to be deleted and click the “Delete” button on the Worksheet Manager toolbar. A confirmation dialog box will be shown. Following confirmation, the selected sheets will be removed from the displayed list.

The sheets are not actually deleted from the workbook until the "Apply" or "OK" button has been clicked.

REORDERING

Sheets can be reordered by sorting, or by clicking the up and down arrows in the toolbar. These operations change the displayed order of the sheets, not their index values. To set the index values to the displayed order, click the "Set Order" button on the Worksheet Manager toolbar.

When the "Apply" or "OK" button has been clicked, the reordering will be applied. This can take a long time, and can also invalidate 3D references.

FIND / REPLACE (RENAMING)

The Find and Replace panel is displayed by clicking the "Find" or "Replace" buttons on the Worksheet Manager toolbar. If the panel is displayed using the "Find" button, the replacement options will be disabled.

The text to be found is entered in the "Find" combo box. Enter a new value, or select an old one from the list. Wildcards using * and ? are accepted.

When the "Replace" controls are activated, the replacement text can be entered or selected in the “Replace with” combo box.
The search can be restricted to the selected worksheets only by checking the "Search selection only" check box.

Case sensitivity can be specified with the "Match case" check box.

Once the "Find Next" button has been clicked and the first match has been found, the Worksheet Manager enters a search state. The user is able to edit the current item, pressing enter to accept the change, and click "Find Next" to move on to the next match. To get out of this state, click the "Reset" button that is located below it.

For replacement operations, click "Replace" to step through the replacements one by one, or "Replace All" to attempt all the replacements.

It is quite simple to specify a replacement that results in a set of names that Excel would not allow, e.g. replace "Sheet?" with "Sheet". In this case the Worksheet Manager will not allow the replacement.

EDITING SHEETS

The "Protected" and "Calculate" properties can be edited directly in the grid.

Other properties are edited in the "Edit" panel below the grid.

The enter/return key can be used to accept the name value.

The sheets are not actually renamed in the workbook until the "Apply" or "OK" button has been clicked.

2.37.2 Type Name for Sheets

This feature enables the user to define the same name locally in each worksheet, and set them to values that indicate the purpose of the worksheets.

Charts and dialog sheets do not support names, so this feature applies only to worksheets and macro sheets.

Typical values might include "Input", "Workings" or "Report".

OAK’s Worksheet Manager makes it very easy. It will do the following:

- List the value of each worksheet’s type name alongside the other data.
- Edit the value, either by selection from a drop-down list of existing values, or by entering new ones.
- Locate an existing type name, by presenting a list of candidates.
AN EXAMPLE

Starting with a workbook with 6 worksheets and 4 charts, we see that no type name values are visible.

Clicking the ellipsis button on the Worksheet Manager toolbar yields the "Specify Name for Display" dialog.

This dialog enables the user to specify the name to be displayed, along with the column title.

To select an existing name, click the ellipsis button beside the Name text box. This displays the "Select Name for Display" dialog.
Using OAK: OAK commands

This dialog lists all the local names in the workbook that have constant values. It lists the values the name contains in various sheets, and the number of sheets the name is present in.

In this example, the name "Purpose" is the most likely candidate, considering that it is present in 6 of the 10 sheets (with 4 charts and 6 worksheets, there are only 6 sheets that the name can be in), and the values it has. However, one cannot always rely on an existing type name to be in every worksheet.

Clicking the OK buttons on both this dialog and the previous, the user is returned to the Worksheet Manager, which now shows the type names both in the grid and the drop-down list.
2.37.3 Worksheet Manager Options

The Worksheet Manager tab on the Options form provides a number of configuration settings for the Worksheet Manager.

These are Calculation, Sheet Type Indicator and input sheet name defaults.

CALCULATION

The Worksheet Manager gives the user the power to easily change the calculation property of a worksheet. This is potentially dangerous, so a variety of options have been provided to safeguard this operation.

Read Only - User cannot change the property value.
Enable Only - User can only enable the property.
Yes/No prompt - User can change the value, but will be presented with a confirmation when turning calculation off.

SHEET TYPE INDICATOR

The Worksheet Manager enables the user to configure a name that is defined locally on each sheet, to indicate the purpose of that sheet. Excel terminology already includes "Sheet Type" to distinguish worksheets, charts, macro sheets and dialogs.
These settings specify the column titles for these types.

The "Named Type" value is the displayed column title for a sheet type defined in a local name in the sheet.

The "Excel Sheet Type" value is the displayed column title for the Excel sheet type, e.g. Worksheet, Chart, etc.

INSERT SHEET NAME DEFAULTS

These are the default names that will be used when the Worksheet Manager is used to create new sheets. The default name will be suffixed with a number. The original names can be reset by clicking the Reset button on the form.

2.38 Options

The Options window has five tabs, User Interface, Colors, OAK Menus, Worksheet Manager and Updates. It is accessed using OAK Development/Review | Help | Options

USER INTERFACE OPTIONS

Use this to change the selection mode of the Copy Literal command.

COLOR OPTIONS

The Colors tab of the OAK options dialog provides access to dialogs that control the colors that are used when preparing maps of the contents of workbooks or worksheets or comparing workbooks, worksheets or ranges.
MAP WORKBOOK COLOR OPTIONS

Selecting the Foreground, Background or Border Color options will bring up a menu that allows the desired color for that particular item to be chosen.

Changing the foreground color will change the color of the symbol in the cell, except for the array option in which case the color of the hatching will be changed.

Selecting Default Colors will change all of the colors back to their original settings.
Using OAK: OAK commands

COMPARE COLOR OPTIONS

Clicking any of the "Set Color" buttons will bring up a menu that allows the desired color for that particular item to be chosen.

OAK MENU OPTIONS

Shortcut Key - Alters the shortcut or accelerator key used to activate the OAK menu.

Ribbon/Menu Selection - Specifies whether to display the menu in the Add-ins tab or the OAK ribbon tabs.

Context Menu Positions - Specifies the location for the OAK context menu.

- Cell menu - accessed by right clicking a range that is not a whole row or column
- Column menu - accessed by right clicking a column.
- Row menu - accessed by right clicking a row
UPDATE OPTIONS

In addition to the Check for Updates window, the frequency of automatic update checking can be configured here.

WORKSHEET MANAGER OPTIONS

These options are for configuring the Worksheet Manager, and are described in the Worksheet Manager Options section.
Scripting OAK
3.1 Introduction to scripting

A distinctive quality of OAK is that it can be scripted, that is, driven by a program prepared by you.

This capability is significant because the market for software that can manipulate spreadsheets falls into two categories.

- There are applications that are used by a single analyst, typically on a single spreadsheet, and which cost a few hundred pounds, dollars or euros. OAK is one example of these.
- And there are much more costly systems, which use centralised databases to track all the spreadsheet activity within a department or a whole enterprise, often to comply with regulatory regimes such as Sarbanes Oxley. Manufacturers of these systems include Prodiance and Cluster Seven.

The ability to direct the facilities of OAK from a simple program blurs the formerly sharp distinction between these categories. Some things that previously could only be done using one of the expensive software applications may be possible with OAK and a few dozen lines of code.

This section describes how to use the scripting facilities of OAK. The examples are given using VBA, but OAK can be driven from C# or other programming languages that can interface with Microsoft’s Component Object Model (COM).

3.2 Getting started in scripting

To get started in scripting OAK:

1. Get into the VBA development environment. Alt+F11 will do this.

2. Use the Tools | References command to find the list of software libraries available for VBA to access.

3. Find the entry for the Operis Analysis Kit VBA Type Library in the library and enable it.

4. Press OK to dismiss the Tools | References dialog.
5 Now press F2. That will bring up the Class Browser. If you filter for Operis_OAK, you will find the various methods and properties listed in this help.

6 Write the following code in a VBA module

```
Option Explicit

Sub Main()
    Dim o As Operis_OAK.IOAKAddIn
    Set o = CreateObject("Operis.OAK.Connect")
    Set o.ExcelApplication = Application

    o.BuildNameDatabase
End Sub
```

7 Run this and the dialog of OAK's Build name database command will appear.
Click OK, and OAK will generate a report of the names that are defined in the currently active workbook. (If, as is likely, that's a new, unused workbook, the report will say that there are no names defined in that workbook.)

8 Change the code a little (changes in bold)

```
Option Explicit

Sub Main()
    Dim o As Operis_OAK.IOAKAPI
    Set o = CreateObject("Operis.OAK.Connect")
    Set o.ExcelApplication = Application
    o.BuildNameDatabase ActiveWorkbook, True, Nothing, False, True, True
End Sub
```

This time, when the code is run, the same report is generated, but without the dialog appearing. The characteristics of the required report are specified in the program as arguments to the BuildNameDatabase command, rather than manually by the user through the dialog.

### 3.3 Making scripting faster

#### EXAMPLE 1: BASIC USE

To introduce OAK scripting as simply as possible, we intentionally provided an example code fragment that had as few lines as possible.

```
Option Explicit

Sub Main()
    Dim o As Operis_OAK.IOAKAddIn
    Set o = CreateObject("Operis.OAK.Connect")
    Set o.ExcelApplication = Application
    o.BuildNameDatabase
End Sub
```
EXAMPLE 2: CACHING THE COM OBJECT

The CreateObject step is relatively expensive. If you are going to use OAK scripting intensively, it is a good plan to cache this item so that the object only gets created once in a session.

```
Option Explicit

Private oakObject As Operis_OAK.IOAKAddIn

Function MyOAKAddIn as Operis_OAK.IOAKAddIn
    If oakObject is Nothing Then
        Set oakObject = CreateObject("Operis.OAK.Connect")
        Set oakObject.ExcelApplication = Application
    End If
    Set MyOAKAddIn = oakObject
End Function

Sub Main()
    MyOAKAddIn.BuildNameDatabase
End Sub
```

The IOAKAPI object can be cached similarly.

```
Option Explicit

Private oakObject As Operis_OAK.IOAKAPI

Function MyOAKAPI as Operis_OAK.IOAKAPI
    If oakObject is Nothing Then
        Set oakObject = CreateObject("Operis.OAK.Connect")
        Set oakObject.ExcelApplication = Application
    End If
    Set MyOAKAPI = oakObject
End Function

Sub Main()
    Dim r as Range
    Set r = MyOAKAPI.SearchFor(Selection, "dog", SearchContentType_Value, True)
    If Not r Is Nothing then r.Select
End Sub
```

EXAMPLE 3: MODULES AND PROPERTIES

Since the OAK COM class implements both the IOAKAPI and IOAKAddIn interfaces, the same cached object can be used for both:

```
Module: OperisAnalysisKit

Option Explicit

Private oakObject As Object
```
Private Function GetOAKObject() As Object
    If oakObject Is Nothing Then
        Set oakObject = CreateObject("Operis.OAK.Connect")
        Set oakObject.ExcelApplication = Application
    End If
    Set GetOAKObject = oakObject
End Function

Public Property Get API() As Operis_OAK.IOAKAPI
    Set API = GetOAKObject
End Property

Public Property Get UI() As Operis_OAK.IOAKAddIn
    Set UI = GetOAKObject
End Property

Module: OtherModule

Option Explicit

Public Sub UseOAK1()
    Dim r As Range
    Set r = OperisAnalysisKit.API.SearchFor(Selection, "dog", SearchContentType_Value, True)
    If Not r Is Nothing Then r.Select
End Sub

Public Sub UseOAK2()
    OperisAnalysisKit.UI.BuildNameDatabase
End Sub

3.4 OAK methods in the object browser

Once the reference to the Operis Analysis Kit VBA Type Library has been added to a VBA project, the properties and methods exposed by OAK are available for inspection using the VBA development environment's Object browser.

1. Press F2 to active Excel's object browsers

2. From the drop down at the top left of the object browser, select Operis_OAK.
3 The object browser will display the interfaces, methods and properties made available by OAK.

SEE ALSO

OAK object reference

3.5 Script example: Finding redundant spreadsheets

Search a typical company's main file server for documents matching *.xls, and a typical result will be a list of spreadsheets numbering tens of thousands. This is a real obstacle to taking spreadsheet inventories, as required by Sarbanes Oxley and other regulatory regimes.

But many of those spreadsheets are merely copies of others. OAK can identify spreadsheets that are not very different from each other.
This routine examines all the workbooks in a particular directory and identifies the one that is most like the active workbook. Unlike a cell by cell comparison, this comparison takes advantages of OAK's ability to align spreadsheets before comparing them, so that spreadsheets that are only slightly dissimilar do not result in the identification of hundreds of unmatched cell references in formulas.

Option Explicit

Sub Main()
    Dim sDirectoryToSearch As String
    Dim sFileBeingConsidered As String, sFileClosest As String
    Dim dDifferencesInFileClosest As Double
    Dim r As ICompareResult, o As Operis.OAK.IOAKAPI
    Dim wbOriginal As Workbook, wbBeingConsidered As Workbook

    Set o = CreateObject("Operis.OAK.Connect")
    Set o.ExcelApplication = Application
    Set wbOriginal = ActiveWorkbook
    dDifferencesInFileClosest = 999999999 ' Assume there will be no more changes than this
    sDirectoryToSearch = Environ("UserProfile") & "\Documents\"
    sFileBeingConsidered = Dir(sDirectoryToSearch & "*.xls")
    While sFileBeingConsidered <> ""
        If sFileBeingConsidered <> wbOriginal.Name Then
            Set wbBeingConsidered = Workbooks.Open(sDirectoryToSearch & "\" & sFileBeingConsidered)
            Set r = o.Compare(wbOriginal, wbBeingConsidered, True, True, True, True, False, False,
                True, False, "A:B", Nothing)
            If r.TotalDifferences < dDifferencesInFileClosest Then
                dDifferencesInFileClosest = r.TotalDifferences
                sFileClosest = sFileBeingConsidered
            End If
            r.ReportBook.Close
            wbBeingConsidered.Close False
            o.UndoCompareModifications wbOriginal
        End If
        sFileBeingConsidered = Dir
    Wend
    MsgBox "Most similar file to '" & wbOriginal.Name & "' is '" & sFileClosest & "'," & vbOKOnly, "OAK comparison"
End Sub

Tip: In this example, the bold True causes a report book to be written, which is then closed by the line also in bold, r.ReportBook.Close. This is to give a pretext to illustrate use of the r.ReportBook method. If the bold True is changed to False, no report book is generated, allowing the bold r.ReportBook.Close line to be omitted. Not having to generate a report workbook would allow this code to run faster.

3.6 Colors in compare and map commands

Some of OAK's API commands for comparing items and mapping have parameters for specifying colors:

CompareWithColors
CompareRangesWithColors

Map

These functions use numbers to specify colors. These are 32-bit integers, passed as type long in VBA.

The following VBA function shows how they can be specified from red, green and blue values in the range 0..255.

Function Win32Color(red As Byte, green As Byte, blue As Byte) As Long

    Win32Color = (CLng(blue) * 256 + green) * 256 + red

End Function
The Desktop Edition of OAK is a standalone application which drives Excel to open
workbooks and perform Map, Summarize and Comparison operations.

Due to memory constraints running OAK as an add-in in 32-bit editions of Excel, users may often find that performing maps, summaries and comparisons on workbooks that take up a lot of memory in Excel when opened, will fail with out of memory errors. The Desktop Edition of OAK is intended to provide a more memory-efficient and robust alternative.

### 4.1 Using the Desktop Edition

Start OAK4 Desktop Edition by locating it under Start | All Programs | Operis | OAK4 and clicking its icon. When started, the application presents the following user interface:

To use OAK4 Desktop Edition, you need to specify the workbook(s). There are two group boxes in the OAK4 Desktop window labeled Workbook 1 and Workbook 2, and each has a text box for the file name, its location, a button to open a file and a button to clear the details. While you can specify a workbook by clicking the Open button and locating the file in the file system, if you already have the file visible on your desktop, you can simply drag the file into the Workbook 1 or Workbook 2 boxes and drop it there.

Once you have specified 1 workbook, the Map and Summarize buttons, which were disabled, become enabled:
Drag and drop another workbook into the Workbook 2 area, or open the file using the Open button in the Workbook 2 area.

Now the details for the 2 workbooks are displayed, and the Compare button is also enabled.

**USING DESKTOP OAK FUNCTIONS**

To use the functions of OAK4 Desktop Edition, simply click the Map, Summarize or Compare button once you have specified the workbook(s) to work on. The same options dialog displayed by the OAK add-in is used to configure the operation.

These operations are described in Section OAK commands:

For Map, refer to Section Map.

For Summarize, refer to Section Summarize.
For Compare, refer to Section Compare.

4.2 History and Purpose

During 1998-2014, OAK was only installed as an add-in in Excel, and since OAK4 it has been a COM add-in which runs in the same process as Excel. In 32-bit Windows, this process cannot have more than approximately 1.3Gb of random access memory (RAM) allocated to it, regardless of how much RAM is installed on the computer.

This limit still applies when OAK is running in 32-bit Excel 2010 to 2016, under 64-bit Windows. The computer could have 16Gb of RAM installed, but 32-bit Excel will still only be able to use around 1.3Gb of it.

Since the adoption of Excel 2010, Operis noticed a large increase in occurrences of OAK failing with out of memory errors. While a great deal of effort has gone into making sure that OAK promptly releases resources that Excel needs to clean up, this has often not been enough.

For this reason, the desktop edition was created. Rather than being an add-in that runs inside Excel and is called by it, it is quite the opposite. OAK Desktop Edition is a standalone application, which runs in its own process. In 64-bit Windows it will run 64-bit, and consequently with the ability to access very much more than 1.3Gb of the computer’s memory, even if the installed edition of Excel is 32-bit. Excel will still be memory-limited in the things it can do, but it will not be burdened with the memory footprint of OAK’s operations as well.

OAK4 Desktop Edition provides a more memory-efficient means of running OAK summaries, maps and comparisons.
Reference material
Basic concepts in Excel
Reference material: Basic concepts in Excel

1.1 Basic concepts in Excel

Certain ideas that underpin Excel are presented here

- mainly, because it is necessary to be familiar with them in order to understand the intent behind some features of OAK and why they might be useful
- also, as a service to the community, because they are useful things for Excel users to know whether or not they choose to use OAK.

Multi-area ranges  Subsets of the cells on a worksheet that are arranged in shapes more elaborate than a rectangle

Used range  The area of each worksheet for which storage is reserved

Cell precedents  The cells that a formula acts on to reach a result

One-Cell rule  A rule that determines which cells commands should act on

Merged cells  A way of formatting a group of cells so that it appears to be a single cell

Arrays  An advanced kind of formula that processes lists

1.2 Multi-area ranges

SELECTING CELLS

One of the commonest actions in Excel is to select a group of cells arranged in a rectangle on a worksheet.

By using the mouse  By using the mouse  By using the keyboard

Everyone knows this way  Not so many people know this way  Not so many people know this way

Click on a cell in one corner of the rectangle  Click on a cell in one corner of the rectangle  Use the arrow keys to navigate to a cell in one corner of the rectangle

**Hold down** the left mouse button  **Hold down** the Shift key  **Hold down** the Shift key

Click on the cell at the diagonally opposite corner of the rectangle  Click on the cell at the diagonally opposite corner of the rectangle  Use the arrow keys to navigate to the cell at the diagonally opposite corner
DESELECTING CELLS

Once you have made a selection of some cells on a worksheet, how can you get rid of the selection? You can deselect the cells in two ways:

- with the mouse: click on any single cell
- with the keyboard: press any of the arrow keys

The rule is simple: **when you move about the spreadsheet**, either by clicking on a cell or using the arrow keys on the keyboard,

- while holding down the mouse button or the Shift key: Excel selects any cells that you pass over
- **without holding down the mouse button or the Shift key**: Excel deselects any previously selected cells.

MORE ELABORATE SELECTING

As we’ve just learned, **when you move about the spreadsheet without holding down the mouse button or the Shift key**, Excel deselects any previously selected cells.

But there are ways to stop that happening.

- With the mouse: Select some cells. Hold the Ctrl key down. Select some other cells.
- With the keyboard or the mouse: Select some cells. Press Shift+F8. Select some other cells.

In both cases, the second group of cells is not selected *instead of* the first. Instead, the second is selected *as well as* the first.

This process can be repeated as necessary to select up multi-area ranges of arbitrary shape.

WHY THIS IS USEFUL

In cell A1 on a blank workbook, enter a formula =C3+C4+D6+E6+F8+F9

Now, while that cell is selected, press Ctrl+[. (Hold the Ctrl key down, and press the left square bracket.) Excel will change the selection from A1 to those cells mentioned in the formula. This is because pressing Ctrl+[ is one way to find the precedents of a cell.
The cells mentioned in the formula don't lie in a simple rectangle, but that presents no obstacle to Excel in selecting them. Excel simply selects one of the multi-area ranges that we have just learned about.

There are several ways to find the precedents of a cell. Ctrl+[^ is just one of them. Another is to use Go To Special.

- Select cell A1, the one in which you have put the formula.
- Press F5, the short cut for Go To
- In the first dialog that appears, press the Special button in the bottom left corner.

- In the second dialog that appears, select Precedents

- Click OK or press Enter.

The cells mentioned in the formula will be selected, as before.

Finding a cell's precedents is just one of the options offered by the Go To Special dialog. They are basic tools in spotting blemishes in spreadsheets. With just a couple of exceptions, all of them will select a multi-area range when appropriate. (The exceptions find Comments and Objects, which float over the grid of cells, so multi-area ranges are not relevant.) In this way Excel's ability to select ranges of arbitrary shape underpins the facilities it provides for checking spreadsheets.
THE ACTIVE CELL

Clicking on a cell, or navigating to it using the arrow keys, instructs Excel to make that cell the Active Cell. It is the contents of the Active Cell that are displayed in the Formula Bar above the worksheet, and it is the cell address of the Active Cell that is displayed in what Microsoft calls the Name Box, to the left of the Formula Bar.

If you select many cells on a worksheet, the Active Cell will be among them. It is differently highlighted from the other cells.

MOVING THE ACTIVE CELL

Once Excel's Go To Special command has selected cells of a particular kind, the commonest next step is to examine the contents of those cells. However Excel only displays in the Formula Bar the contents of one cell, the Active Cell. And as we have already learned, **when you move about the spreadsheet without holding down the mouse button or the Shift key, Excel deselects any previously selected cells.**

So any attempt to navigate around the worksheet, to inspect cells within the selection other than the currently Active Cell, risks losing that selection.

The trick for resolving this conundrum is to press the Tab key. The Tab key will visit each cell within a selection in turn, and it will do it while keeping the relevant cells selected. Shift+Tab does the same thing, but travels in the opposite direction.

In fact there are seven key combinations that move the active cell within a range.

- Tab, as we have just seen, moves from left to right within the selection
- Shift+Tab, moves from right to left within the selection
- Enter, moves downwards within the selection
- Shift+Enter, moves upwards within the selection
- Ctrl+. (full stop, or period), moves clockwise among the corners of the area containing the Active Cell
- Ctrl+Alt+Right moves to the top left corner of the next area of a multi-area selection
- Ctrl+Alt+Left moves to the top left corner of the next area of a multi-area selection.

WHY THIS IS USEFUL

Excel's Go To | Special command can be instructed to identify interesting cells, and these can then be visited in turn by pressing the Tab key, and their contents examined. This is a basic technique in examining a spreadsheet.

OAK's Search Selection commands mimic this behaviour.
1.3 Seven ways to identify a cell's precedents

Understanding which cells a formula refers to, and coming to a view on whether they are the correct cells in the circumstances, is a central activity in checking a spreadsheet. Excel offers about seven ways of determining the precedents of a cell, and they are all slightly different.

INSPECTION

The most obvious method for checking a formula’s precedents is to examine it and to navigate to any cells that it mentions, one after the other.

GO TO SPECIAL

One of the options of Excel's Home | Find & Select | Go To Special is Precedents.

Activating this will

- either, select a multi-area range consisting of those cells that are mentioned in any formula in the selection, so long as they are on the same worksheet
- or, report that no such precedents exist.

This command dates back to the very earliest versions of Excel, which were too primitive to have any notion of workbooks that might contain several worksheets. If a formula mentions a cell on another worksheet, with a formula such as =Sheet2!A1, Excel will ignore the fact. A selection of one or more formulas that don’t mention any cells on the same worksheet will be reported as having no precedents even though they do refer to numerous cells on other worksheets.

CTRL+[ (LEFT SQUARE BRACKET)

The Ctrl+[ key combination works very like Go To | Special | Precedents. In fact, the actions are identical in many circumstances. But there are two situations in which Ctrl+[ will act differently.
• If the first reference in the formula of the active cell is to a cell or range on another worksheet, Excel will make that the selection, and ignore all the other precedents in that active cell or any other selected cells.

• If there are no precedents found, Ctrl+{ won't complain. It will do nothing.

DOUBLE CLICK

Older versions of Excel allowed the precedents of a cell to be selected if the cell was double clicked. More recent versions use double clicking on the cell to mean you want to edit its contents. But the old behaviour can be restored by turning off the "Edit directly in cell" feature. This option is reached with Office button | Excel options | Advanced | Allow editing directly in cells.

The action is identical to Ctrl+[, except that it applies to the cell double clicked on only (Ctrl-[ will show the precedents of all cells selected if there is more than one.)

FORMULAS | FORMULA AUDITING | TRACE PRECEDENTS

The Trace Precedents command draws

• blue arrows to indicate a linkage between the active cell and any other cells it refers to on the active workbook. Navigation is possible between the cells by clicking on the arrows.

• a black arrow, leading to an icon representing other worksheets, which can be double clicked to see what references are present to other worksheets or workbooks and to navigate to them.

FORMULAS | FORMULA AUDITING | EVALUATE FORMULA

The Evaluate Formula command brings up a dialog through which a formula can be probed interactively.

• Good points: It can step through the precedent hierarchy of a calculation, and understands row and column matching.

• Bad points: It evaluates lookup formulae to values rather than references, and insists on visiting terms in a prescribed order, left to right through the expression text, with no opportunity offered for backtracking.

GO TO WHILE EDITING

Using Excel's Go To command is such a basic action that this section of the help starts by recapping it. The shortcut for it, F5, is well known. But what isn't so well known is that you can press F5 while editing a formula. If the portion of the formula being edited is a reference, then F5 will navigate to the reference and adjust the formula.

For example,
Put the number 7 into cell A1 of a blank worksheet.

- Put =6*INDEX(A:A,1) into another cell. The answer will be 42, because INDEX(A:A,1) evaluates to the first cell in row A, in other words, A1.
- Start editing the cell and select the last final characters, the ones that read "INDEX(A:A,1)".
- Now press F5. Excel will navigate to cell A1, scrolling the worksheet if necessary until A1 is brought into view, and indicate the cell with a border of animated dashes.
- The text of the formula will change. Rather than =6*INDEX(A:A,1), it will read =6*A1
- There are circumstances when debugging a formula that you may like the change Excel has made in your formula, since it is so much easier to read. In such a case you would press Enter to keep the revised formula. But most times, you would press Ctrl+Z (Undo), to go back to the original formula, or Esc to abandon the formula edit altogether.

You won't be able to see what's in A1, since Excel is busy displaying the formula that you are editing. But you will see where it is, and what labels surround it.

WHY THIS IS USEFUL

Understanding which cells a formula refers to, and coming to a view on whether they are the correct cells in the circumstances, is a central activity in checking a spreadsheet. That's why this help offered finding cell precedents as the most relevant example of why Excel's ability to select ranges that are of arbitrary shapes is interesting.

1.4 Used range

In Excel, there is a concept known as the "used range"; the region on each individual worksheet which Excel thinks is being used. The larger the used range, the larger your file size and the slower your workbook calculation speed.

You can identify this region on a worksheet yourself by pressing the Ctrl+End keys. This takes you to the cell in the bottom right-hand corner of the used range, referred to as the last cell. Thus the used range on your worksheet is defined as the rectangular region from cell A1 to the last cell.

The last cell can be empty because it is determined by finding the intersection of the maximum used column and the maximum used row.

Usually the last cell is at, or only a few rows or columns beyond, the final cell which actually contains formulas or constants. This means that the used range is no larger than, or only marginally larger than, the minimum size it could be.

However, there are times when the last cell, as defined by Excel, is far beyond the final cell which contains information, making the used range much larger than it should be.
One of the items which is listed on the OAK Workbook Summary report is the last cell location. This tells you which worksheets in your workbook have used ranges which are much larger than the region on the worksheet which actually contains non-blank cells.

WHY THIS IS USEFUL

If the used range is unnecessarily large, the size of your XLS file will be larger than it should be and your workbook calculates more slowly than it should. Even navigating around the worksheets can be sluggish.

To reduce the used range of a worksheet to the correct size once it has grown very large:

1. Select the excess rows or columns on your worksheet;
2. Choose the Excel Home | Cells | Delete command (shortcut Ctrl+minus) 
3. Save the workbook as a new file; use the Excel Office button | Save As command;
4. Close the workbook;
5. Open the new file you have just created and select the appropriate worksheet;
6. Press Ctrl+End to test the location of the last cell.

If you import a Lotus 1-2-3 spreadsheet into Excel, you may find that the used range is much larger than it should be. Following the steps outlined above will often resolve the problem.

1.5 One-Cell Rule

A number of OAK commands follow what Operis calls the One-Cell Rule. It is a rule that some built-in Excel commands also follow.

Here’s the rule:

If you have just one cell selected when you perform an action, the action applies to the whole of the active worksheet. But if you have two or more cells selected, the action is confined to the selected cells.

To understand this, try the following example:

1. Select half a dozen or so cells on a blank worksheet;
2. Type the word “dog”; hold Ctrl down when you press ENTER, so that the word is typed into all of the selected cells;
3. Now select any one cell on the worksheet;
4. Use the Home | Editing | Find & Select | Replace command to change all occurrences of the word “dog” to “cat” (press the ‘Replace All’ button on the Replace dialog); notice that all the cells with text in them change;
5. Now select a portion of the worksheet that includes some, but not all, of the cells that now say “cat”;
6. Use the Excel Home | Editing | Find & Select | Replace command to change all occurrences of the word “cat” to “mouse”; notice that this time, only the cells that were inside the selected area are changed.

Four Excel commands work in this way. They are:

- Edit Home | Editing | Find & Select | Replace, as we have just seen;
- Edit Home | Editing | Find & Select | Find, which is related to it;
- Home | Find & Select | Goto Special (at least for most of its options);
- Formulas | Defined names | Defined name | Apply Names.

WHY THIS IS USEFUL

The following OAK commands intentionally work in the same way:

- OAK Development | Names | Apply;
- all of the commands on the OAK Development/Review | Search menu.

There are two OAK commands that notably do not follow this rule:

- OAK Development | Names | Delete;
- OAK Development | Names | Localize.

1.6 Merged cells

Excel allows any contiguous, rectangular selection of cells to be merged into a single cell. The merged cell contains only the value or formula which originally existed in the upper, left-most cell in the selection. If any of the other cells contained information, it is discarded when they are merged.

There are good things to be said about the ability to merge cells. It allows cells to be merged vertically as well as horizontally; that can be handy when producing headings down the left side of a table that relate to several rows. And it harmonizes the facilities of Excel with that of Word’s tables.

But there are many practical reasons to dislike merged cells.

- Highlight a rectangular range of cells that includes some merged cells, and the selection will automatically be enlarged so that it includes the whole of the merged area. It is impossible to select a smaller area.
- You can select an entire column or row by clicking on its header. But if the column or row intersects any merged cells, those cells get selected too. That can be tiresome; in particular, it stops the Insert rows and Insert columns commands from working. (OAK provides its own Insert and Delete commands which gets round this.)
- You can select an entire column or row with the keyboard: Ctrl+Space selects the column, Shift+Space the row. But if the column or row intersects any merged cells, those
cells get selected too, along with the entire columns or rows that they in turn intersect. In most circumstances that renders the ability to select rows and columns with the keyboard unusable. This is particularly confusing if the merged cells happen to be hidden.

- Text that won't fit in a cell is usually able to spill out into its neighbors, so long as they are unoccupied. In a merged cell, however, the text is truncated.

- Unlike every other cell formatting option, the merging of cells is not simply a characteristic of a format style; it actually changes the regular grid structure of your spreadsheet.

Avoiding using the merged cell facility is no hardship, since another command is available which dates back to earlier versions of Excel that didn't support cell merging. It is the option "center across selection", which can be reached from the Alignment option offered by the Format Cells command (shortcut Ctrl+1). It produces a cosmetic effect that is similar in many cases to merging cells, without any of the drawbacks. (Exception: merging cells works horizontally and vertically. Center across selection is useful for horizontal groups of cells only.)

We recommend users of Excel to favor the center across cells facility over merging cells, and actively to reverse any cell merging in existing spreadsheets. The only problem is that the cells that have been merged in a spreadsheet are hard to find; there's nothing to identify them. That's why one of the search selection options offered by OAK is one that picks out the cells that have been merged in a worksheet.

1.7 Arrays

Excel has a special kind of advanced formula known as an array. These are different from conventional formulas in that a single array formula can return more than one result. This powerful feature can be very useful, but it needs to be used with skill and care.

In order to create an array formula, you must do the following:

1. Select the range of cells which you wish to contain the array formula;
2. Type in the formula, but do not press Enter;
3. Press Ctrl+Shift+Enter (instead of Enter) to turn the formula into an array formula in all of the selected cells.

To understand this, try the following simple example on a blank worksheet (and see the screenshot on the next page):

1. On a blank worksheet, type decimal figures into cells B4 to G4, making sure that some of the numbers are positive and some are negative;
2. Select a cell in a different row;
3. Type in the formula: =min(if(B4:G4<0,0,B4:G4)), but do not press Enter;
4. Press Ctrl+Shift+Enter to create the array formula.
Note that the formula is surrounded by braces: `{ }`. This identifies it as an array formula. This particular formula finds the minimum value in the selected range of numbers which is greater than zero. The IF statement returns all of the values in the range which are greater than zero and then the MIN function finds the minimum of those. The key to this formula working as it does is that as an array formula, the IF statement can return more than one value.

When using arrays, there are a few important points to keep in mind.

- Excel will not allow you to modify the formula in one of the array formula cells without modifying them all. This is because all of the cells in an array formula must by definition contain exactly the same formula.

- If you select the entire range containing the array formula, you can cut the selection and paste it anywhere else on the worksheet. In other words, the array does not have to be lined up with the cells it references.

- Finally, note that Excel ensures that data tables recalculate by placing them in a special kind of array formula.
Excel Names
Overview of Excel names

Everyone who has used a spreadsheet for a non-trivial purpose knows that results are calculated using a formula notation in which cells are identified with coordinates, as in =A5-B7.

What is much less widely known is that it is possible to associate meaningful names with those cells, and use the names in the formulas, as in =Revenue-Costs.

Opinion is more divided on whether it's a good idea for formulas to identify their inputs by names rather than coordinates in formulas than by any other topic in the field of developing spreadsheet models.

There are some, Operis among them, who like names and use them extensively in their spreadsheets. OAK has a number of facilities that are valuable to people with this outlook.

There are other practitioners who dislike names and discourage their use in spreadsheets. OAK has facilities that they will find valuable also.

The majority of spreadsheet users have no opinion on the subject, because they know next to nothing of the use of names in spreadsheets. Since quite a proportion of OAK is devoted to extending Excel's functionality for handling names, this help has a fifteen step tutorial that explains the naming facilities offered by Excel, and what OAK does to enhance them.

If you use Excel names

Opinion is divided between spreadsheet users who value Excel's ability to express formulas in terms of meaningful names, such as =Revenue-Costs, and others who prefer the traditional coordinate notation, such as =D5-D7.

If you like using names, OAK provides a number of facilities that may help you.

- You can change your mind about the names that you allocate to quantities using the select OAK Development | Names | Redefine and OAK Development | Names | Recreate commands.
- You can quickly remove any names that have become broken with the OAK Development | Names | Remove #REF! command.
- You can recover quickly from accidentally filling the name space with unwanted names through mistaken use of Excel's Create Names command, by using the OAK Development | Names | Delete command.
- You can change the scope of a quantity of names added by Excel's Create Names command from global (associated with a workbook) to local (associated with a worksheet) by using the OAK Development | Names | Localize command.

Though you may like using names, you will sooner or later encounter a client or a partner who does not like the things. OAK provides a number of facilities that may help you.
You can direct him to this free help, which gives lessons in the use of names, so that he may more easily understand your work.

You can provide a list of all the names in a workbook, so that your counterparty can understand what they refer to, using the OAK Review | Names | Build name database command.

You can provide a map of the workbook, which will show what parts of it have names associated with them, using the OAK | Review | Map command.

If all else fails, you can develop a workbook using names, and then strip them out before dispatching the spreadsheet to a name-phobic client or partner, returning the formulas to the coordinate notation widely associated with spreadsheets, using the OAK Development | Names | Deapply names command.

2.3 If you avoid Excel names

Opinion is divided between spreadsheet users who value Excel's ability to express formulas in terms of meaningful names, such as =Revenue-Costs, and others who prefer the traditional coordinate notation, such as =D5-D7.

If you don't use names in your own work, in time you will be confronted by a spreadsheet that does make use of them. OAK provides a number of facilities that may help you.

- You can get a list of all the names in a workbook, so that you can understand what they refer to, using the OAK Review | Names | Build name database command.

- You can prepare a map of the workbook, which will show what parts of it have names associated with them, using the OAK | Review | Map command.

- You can identify some names and have OAK remove mention of them from all or part of a worksheet, restating any formulas that use them are rewritten in conventional spreadsheet coordinate notation with the OAK Development | Names | Deapply names command.

- You can identify a formula and have OAK remove mention in it of any names, restating the formula in conventional spreadsheet coordinate notation with the OAK Development | Formula | Optimize command.

- You can get OAK to deliver reports setting out reconstructions of calculations which restate what is going on as simply as possible, without using any names.

2.4 Names tutorial A: Getting started

EXCEL'S GOTO COMMAND.

Imagine you load a large worksheet loaded into Excel, and find the cursor in the top left cell, A1. You wish to examine a calculation that you know to be somewhere near row 3000. How are you going to get there?

There are, of course, several ways to get there.
You could press the Down arrow key 2999 times.

You could press and hold the Down arrow key and leave your computer to do the repeating, until the spreadsheet scrolls to where you want to be.

You could do something similar with the PgDn key, which would move rather faster.

You could place the mouse pointer over the slider in the vertical scroll bar, and move it down until Excel indicates that you have reached row 3000.

All these ways work, but there is a faster way: use Excel's Goto command. It can be activated

- from the Excel ribbon: Home | Editing | Find & Select | Go To
- by using a shortcut that dates back to the dawn of the PC: the F5 key
- by using a shortcut that has its origins in Apple Macintosh software: Ctrl+G.

Any of these work, but the one we will use in this tutorial is the F5 key.

To reach row 3000 of the spreadsheet, you simply press F5, and type A3000 into the dialog that then appears.

Dismiss the dialog by clicking OK, or pressing Enter, and you are immediately at the desired spot.

### 2.5 Names tutorial B: Defining names

**ASSOCIATING NAMES WITH CELLS USING EXCEL’S DEFINE NAME COMMAND**

1. To get started with this lesson, load the example spreadsheet "Names Tutorial B defining names.xls". Instructions on loading example spreadsheets are found in the Example Spreadsheets section of this help.
2 Click on cell B1, which contains 200.

3 Use Excel's Formulas | Defined names | Define name command. A dialog will appear.

Excel has already filled in the several elements of the dialog.

We are on cell B1, so it has prepopulated the lower, Refers to field with that cell address.

It has noticed that immediately next to the select cell is another cell which has a piece of text in it. It has prepopulated the upper, Name field with that text, on the basis that there is a reasonable chance that it might serve as a suitable name to give the selected cell.

The word Sales is shaded, showing that it is selected, so that if we disagree with Excel's guess that this might be a suitable thing to call the cell we have selected, we can enter something we prefer, and the guess will be overtyped immediately.

Here, however, we think that Sales is a good description for the 200 quantity; that is why we put the label there. So we can accept Excel's proposals by pressing Enter, or clicking on the OK button, to dismiss the dialog box.

4 Now try press F5, which we learned in Lesson A was the shortcut for Goto.

When we used the Goto key in Lesson A, on a blank worksheet, there were no entries in this dialog. But now there is an entry: one for Sales, the name we defined a moment ago.
5 Now select cell B2, the one that contains 115. Try Formulas | Defined names | Define name again. As before, Excel prepopulates the dialog:

- the address of the selected cell, B2, is at the bottom, in the Refers to field
- Costs, the contents of a neighboring cell that appeared to offer a plausible candidate for a name, is at the top in the Name field, preselected in case we want to type over it with something we prefer.

6 Again, we can accept Excel's suggestions by clicking OK, or pressing Enter, to dismiss the dialog.

7 This time, the F5 key shows two names, listed alphabetically.
8 Try choosing either of the entries, and clicking OK or pressing Enter to see what happens.

9 Try scrolling far away from the top left corner, then pressing F5 to bring back the Goto dialog, and selecting Sales or Costs and clicking on OK. Excel will speedily select the chosen cell.

2.6 Names tutorial C: Using names in formulas

HOW TO USE NAMES IN FORMULAS.

We have seen that when you press F5, Excel displays its Goto dialog. You can identify a cell you would like to visit by

- either, typing a cell reference, such as B2
- or, typing or selecting a name that you have defined already, such as Costs.

What we will learn now is that the same is true when writing formulas. You can specify the cells that participate in the calculation by

- either, typing a cell reference, such as B2
- or, typing or selecting a name that you have defined already, such as Costs.

Assuming you have completed the exercise in lesson B, your active worksheet will contain a cell named Sales and a cell named Costs. You will also have a cell calculating Profit.

- The profit cell is the one showing the value 85
- Its formula is =B1-B2.

In a blank cell, type a new formula. Type =Sales-Costs. You will find that you get the value 85 again.
In interpreting this formula, Excel has
- seen that we have used the name Sales
- checked its definition
- seen that it refers to cell $B$1
- taken us to mean =$B$1-Costs.

It has then done a similar substitution for Costs, to get a formula =$B$1-$B$2 which, with the exception of four dollar signs, is identical to the original version of the Profit formula.

Tip: The presence or absence of dollar signs makes no difference to how a formula is evaluated. It only makes a difference to Excel's action when Excel the formula is copied into other cells.

The names we are using here are easy to type, being only five letters long. Sometimes, names may be longer and more convoluted. Rather than typing them ourselves, and perhaps spelling them incorrectly, we can get Excel to type them for us.

In another blank cell
- start to type a new formula by typing =
- press the F3 key
- from the list in the dialog that appears, choose Sales, then click on OK or press Enter
- type - (the minus key, because we wish to perform a subtraction)
- press the F3 key
- from the list in the dialog that appears, choose Costs, then click on OK or press Enter
- press Enter to finalize entry of the formula.

Again, the result will be 85.
F3 is the short cut for the Excel command Formula | Defined names | Use in formula | Paste names. It is similar to the Goto command which we've already practiced, in that it presents a dialog displaying list of names that have defined already. However, the circumstances when it is useful, and the action when it is used, are rather different.

<table>
<thead>
<tr>
<th>Short cut</th>
<th>F5</th>
<th>F3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command</td>
<td>Home</td>
<td>Editing</td>
</tr>
<tr>
<td>Normally pressed</td>
<td>When not in edit mode</td>
<td>When in edit mode, to enter a formula</td>
</tr>
<tr>
<td>Action</td>
<td>Selects the cells that are associated with the name.</td>
<td>Pastes the name selected by the user into the formula being constructed.</td>
</tr>
</tbody>
</table>

**Dialog**

![Go To Dialog](image1.png)

![Paste Name Dialog](image2.png)

### 2.7 Names tutorial D: Names adjustment

**EXCEL ADJUSTS THE DEFINITION OF NAMES AUTOMATICALLY.**

1. Assuming you have completed the exercise in lesson C, you will have a cell named Sales and a cell named Costs.

   You can check the definitions of these names.

   - Use Excel's Insert Names Define command.
   - You will see entries for the Sales and Costs names that you have defined already.
   - Click on the entry for Sales. You will see its name in the text box at the top, the Name field, and the address of the cell it is associated with in the text box at the bottom, the Refers to field. That cell is Sheet1!$B$1.
   - Click on the entry for Costs. Now its details are shown in the top and bottom boxes, and its cell is Sheet1!$B$2.
2 You will also have three cells containing versions of the profit calculation, differently entered but all showing the same value, 85. One of them contains the formula =B1-B2.

3 Insert a few rows at the top of the spreadsheet, so that all the non-empty cells move down the worksheet.

4 Take another look at that profit formula. It won't say =B1-B2 any more, because you have moved what were B1 and B2. As even a beginner with spreadsheets knows, Excel adjusts these formulas in these circumstances, and the formula carries on showing 85.

5 Two other cells contain versions of the profit calculation. They express it in names, showing =Sales-Costs. And they also show 85, just like the first formula.

6 We can see why this is by using the Formulas | Defined names | Define name command again and inspecting the definitions of Sales and Costs. You'll find they too have been updated, to reflect the changed position of these two cells, just as the coordinates in the Profit formula were.

7 You can confirm this adjustment in another way. If you press F5, the Goto command, and choose either Sales or Costs from the dialog that appears, you will be taken to the selected cell at its new location.

No matter how much you move the cells around on the worksheet, by cutting and pasting them somewhere else, by selecting them and dragging them into a new place on the worksheet, or by inserting or deleting rows or columns, Excel will keep track of their new position and updates the coordinate references within the name definitions as necessary, just as it updates the coordinate references within all the formulas.

This is really useful. You might recall that your Cost estimate is in cell B2, but after a worksheet has been developed by you and, perhaps, some colleagues, over several months, it might have made its way to cell K937. How would you know? How could you remember? You don't have to. Just give it a meaningful name, and Excel will track it for you. All you have to do to return to it at any later date is press F5 to activate the Goto command, type Costs or choose it from the list of names that is offered, and click OK or press Enter.

2.8 Names tutorial E: Apply names

EXCEL'S APPLY NAMES COMMAND.

1 If you have followed the lessons so far, you will have a spreadsheet containing

   • a cell which you have named Sales
   • a cell which you have named Costs
   • three cells which calculate profit. One does so using traditional coordinate references, and the others do so using names, =Sales-Costs
2. Select the cell which calculates profit using coordinate references. (It was at cell B3, but is likely somewhere else now as we moved things around in Lesson D.)

3. Now invoke Excel's Formulas | Defined names | Defined name | Apply Names command. Like the F5 (Goto) and F3 (Formula | Defined names | Use in formula | Paste names) commands, this one offers a dialog that presents a list of the names defined in the workbook.

4. Choose the entry for Sales and click OK or press Enter to dismiss the dialog.

5. Now examine the coordinate formula. A moment ago it said =B1-B2 (or similar; the actual references may be different as we moved the cells around). Now it will say =Sales-B2 (or similar)

6. Repeat the process, this time applying the name Costs. The formula will be transformed to =Sales-Costs.

By this means we have discovered that the action of the Apply names command is to go through a portion of a spreadsheet looking for opportunities to replace coordinate references in formulas with names.

THE APPLY NAMES COMMAND HAS TWO USES

The obvious use for Apply names is that it leaves us free to enter formulas, and define names, in arbitrary orders. We don't have to think of all the names before we start a spreadsheet and define them in order for them to be available for us to use. We can write a spreadsheet using traditional coordinate formulas, decide to define a few names, and then have Excel do the hard work of adapting the formulas that are already in place to use the just-specified names.

The less obvious use is that writing a formula in coordinate notation, defining some names, and applying those names is great for learning. Sometimes Excel will replace the coordinates in a formula with names and sometimes it won't. Working out why it behaves as it does is much the best way of understanding how Excel interprets names.

Tip: The golden rule is that Apply names never makes a substitution that would cause the meaning of a formula to change. Every time that you expect Excel to substitute a name for some coordinates in your formula, and it doesn't, is an opportunity to revisit your understanding of what the formula would have meant had the substitution happened, and why it is not the same as what you have written in coordinate form.

2.9 Names tutorial F: Names that identify several cells

NAMES ASSOCIATED WITH MORE THAN ONE CELL.
1 To get started with this lesson, load the example spreadsheet "Names Tutorial F Multi cell names.xls". Instructions on loading example spreadsheets are found in Example Spreadsheets section of this help.

This spreadsheet is different from the examples earlier in this tutorial, in that rather than having a single cell for Sales, and another one for Costs, we have five years of them. The sales are calculated by marking up the costs by 20%.

2 Highlight the five costs figures.

3 Use Excel's Insert Names Define command. As it did in Lesson B with single cells, a dialog box will appear, which Excel will have prepopulated with

   - the address of the selected cells (the five Costs figures) in the lower text box, the RefersTo field
   - the word Costs in the upper text box, the Name field.

Excel offers the word Costs because it is located in a cell near to the selected cells and appears to be a plausible candidate for the name we'd like now to create. But Excel selects the name, so that if its choice isn't what we had in mind, we can conveniently overtype it.

4 In this case, Costs will do fine as a name, so we can accept Excel's proposal and dismiss the dialog by clicking OK or pressing Enter.

5 Examine the bottom line, which calculates what we have to charge for our product if we are to add a 20% markup to the costs. See that it does so using a simple formula expressed in coordinates.
6 We carefully introduced Excel's Apply names command in Lesson E, because seeing what it does is the best way to understand how Excel's names work. Try using it now.

- Click on any one cell. (This is necessary because Apply names follows the One-cell rule. It won't do anything useful if we leave the selection at the five costs numbers, since they don't contain any formulas that Apply name might act on.)
- Activate Excel's Formulas | Defined names | Defined name | Apply Names command.
- In the dialog that appears, select the entry for the name we have created, Costs.
- Dismiss the dialog by clicking OK or pressing Enter.

7 Now examine the bottom line, and see that the Sales calculation has changed so that it now reads \( =\text{Costs} \times 120\% \)

HOW EXCEL INTERPRETS THIS FORMULA

The first of these Sales formulas is in cell B7. When Excel needs to evaluate the formula in that cell

- it sees that the formula is \( =\text{Costs} \times 120\% \)
- it looks up the name Costs, and sees that it is defined as \( $B3:F3 \)
- It substitutes this reference in place of the name, making the formula \( = $B3:F3 \times 120\% \)
But now Excel has a problem. It is trying to evaluate one formula, B7, but it has been given five cells, B3:F3. What is it to do? We've no hope of fitting the five numbers into one cell.

Many spreadsheet programs would react to this situation by showing an error (though that's changing*). Excel does something more useful. It solves the problem that it has more inputs than are needed for a single-cell result by doing something pragmatic: it makes a choice of one of the cells from among the ones specified.

The rule used to make this choice is is simple: Excel picks out the cell that is in the same column as the calculation. Our calculation is in cell B7; so of the five cells, B3:F3, the one that it will involve in this calculation is B3.

Subsequent Sales formulas, in cells C7, D7, etc, do the same thing; Excel matches columns so that their formulas act on cells C3, D3, and so on.

* Open Office, Gnumeric, Google spreadsheets and the latest version of MS-Works now all support coordinate matching, no doubt to achieve strong compatibility with Excel. The major package that does not is the once dominant Lotus 1-2-3.

2.10 Names tutorial G: Create names

A QUICK WAY TO CREATE LOTS OF NAMES

So far in this tutorial, each time we have have wanted to associate a name with some cells, we have used the Formulas | Defined names | Define name command. We have needed to invoke that command once for every name we have added.

The spreadsheet that we use as our example in this lesson has have 5 lines of numbers. We could associate useful names with each line by issuing the Formulas | Defined names | Define name command five times, once for each line. That works perfectly well, but we are going to learn a quicker way to do it.

1 To get started with this lesson, load the example spreadsheet "Names tutorial G Create names.xls". Instructions on loading example spreadsheets are found in Example Spreadsheets section of this help.
2 Once the spreadsheet has loaded into Excel, press the F5 key. As we learned in Lesson A, that is the shortcut for Goto, and brings up a dialog.

Satisfy yourself that the dialog has no entries for names defined in this workbook.

3 Select the cells A1:F11.

Use Excel's Formulas | Defined Names | Create from selection command.

A dialog will appear.

Excel may have preselected the option Left column. If it has not, do so yourself.

4 Accept the settings displayed in the dialog by clicking OK or pressing Enter.

5 Press the F5 key again. See that Excel has now added names corresponding to each of the lines.
6 Try choosing one of the names and pressing OK. You will find that Excel will select the numbers next to the label coinciding with the chosen name.

SOME THINGS TO UNDERSTAND

Both the Formulas | Defined names | Define name and Formulas | Defined Names | Create from selection commands add names to the workbook, but their action is slightly different.

- *Formulas | Defined names | Define name* is the more general command. Anything that can be done to a name, it can do, including deleting unwanted names, and advanced actions that we haven't learned about yet. But it can only perform these actions on one name at a time. That's why it is *Define Name*, in the singular.

- *Formulas | Defined Names | Create from selection* is much more limited. It can only make names that are one row high or one column wide. But it has one big advantage, as we have just seen: it can make lots of names in one action. That's why it is *Create Names*, in the plural.

- When we made names in earlier lessons, we used *Formulas | Defined names | Define name*. The names that resulted were exactly the shape of the area we selected immediately before invoking the commands.

- When we made names in this lesson, we used *Formulas | Defined Names | Create from selection*. The names that resulted were not the same shape as the area we had selected. They are each only one row high, though we selected nine rows. And though we included columns A:F in our selection, the names only cover columns B:F.

- When the *Formulas | Defined names | Define name* command is invoked, Excel displays a dialog which it populates with its suggestion for a name, taken from a cell near to the selection that has plausible looking contents. But if you don't like the suggestion, you can overtype it. The *Formulas | Defined Names | Create from selection* command takes names from the cells specified (here, the left column of our selection, column A), and offers you no option to change them.
If you choose to use many names in your models, the ability of Formulas | Defined Names | Create from selection to define a lot of them in one action will save time compared with the ability of Formulas | Defined names | Define name to define only one at a time.

2.11 Names tutorial H: Useful conventions

PROBLEM

To get started with this lesson, load the example spreadsheet "Names tutorial H Conventions with names.xls". Instructions on loading example spreadsheets are found in the Example Spreadsheets section of this help.

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>2001</th>
<th>2002</th>
<th>2003</th>
<th>2004</th>
<th>2005</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Revenue</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Revenue</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>dogs</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>cats</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>fish</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>parrots</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>costs</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>dogs</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>cats</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>fish</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>parrots</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

We’d like to give names to the different lines of numbers. How shall we do it?

1 One way is to use the Formulas | Defined Names | Create from selection command that we have just learned about.

If we highlight cells A1:G13, and use the Formulas | Defined Names | Create from selection command specifying that the names are to be taken from the labels in the left column, we will give the name Revenue to row 3, and Costs to row 9. But those lines have no numbers on them, while the lines that do have numbers remain unnamed. So that’s not much good.

2 If we highlight cells B1:G13, and use Formulas | Defined Names | Create from selection, Excel will complain.

The reason is that Excel works down the selected rows, allocating the name dog to row 4, cat to row 5, fish to row 6 and parrot to row 7. Then it gets to row 10, and finds that
it is being asked to allocate the the name dog again. At this point it pauses for instructions.

Depending on the answer to Excel's question, the name dog will be associated with rows 4 or 10; but not both. One line will go unnamed. The same is true for the other three names on offer.

3 We could alter the labels in columns A and B so that contained some unique names for each of the rows. But to those who think that they do a good job of communicating the intent of the spreadsheet as they are, this would be to compromise their readability.

4 Given that we are not having much luck with Formulas | Defined Names | Create from selection, we could use Formulas | Defined names | Define name. We could highlight cells C4:G4 first, then invoke that command. Excel would propose dog as the name, but we could overtype that with something like SalesOfDogs. This would work, but require that we select each of the eight rows one at a time, and go through the naming process separately for each of them.

SOLUTION

One recommendable way forward is to put the intended names down the right side of the spreadsheet.

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>2001</td>
<td>2002</td>
<td>2003</td>
<td>2004</td>
<td>2005</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Revenue</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>dogs</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>SalesDogs</td>
</tr>
<tr>
<td>5</td>
<td>cats</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>SalesCats</td>
</tr>
<tr>
<td>6</td>
<td>fish</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>SalesFish</td>
</tr>
<tr>
<td>7</td>
<td>parrots</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>SalesParrots</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Costs</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>dogs</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>CostsDogs</td>
</tr>
<tr>
<td>11</td>
<td>cats</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>CostsCats</td>
</tr>
<tr>
<td>12</td>
<td>fish</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>CostsFish</td>
</tr>
<tr>
<td>13</td>
<td>parrots</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>CostsParrots</td>
</tr>
</tbody>
</table>

Then we can select cells C4:H13 (the numbers plus the labels on the right) and use Formulas | Defined Names | Create from selection, this time in the right column.
By devoting the right side of the spreadsheet to names, we attain a layout that is self-documenting (what a row is called is written at its right side) and allows the names to be created in a single action. At the same time the left side can be arranged to make the purpose of the spreadsheet clear to human readers, without any of the compromise that would be needed if the two functions were merged, as they were in steps 1 and 2 above.

**CHOICE OF NAMES**

Remember those shortcuts we learned early on? F5 is Goto; F3 is for pasting names into a formula more accurately than one can type them. And we've also met Apply names. Every one of these commands lists the names in a workbook in alphabetical order. The convention Operis follows is that names are chosen so that related items are near each other in these sorted lists. So here, we have four kinds of Costs, and by calling them CostsDogs, CostsCats and so on, we ensure that they are to be found by each other when presented alphabetically. The noun (Revenue or Costs) goes first, then the qualifier (Dogs, Cats etc). It's a bit like the French preference to put adjectives after nouns (La Maison Blanche) in contrast with the opposite order in English (The White House).

Whether you follow this convention or another, the valuable thing is that you have one, and that it is followed by all members of a team that has to work on each other's spreadsheets.

**SUMMARY**

If you name a cell or a range of cells, document the fact in a cell nearby.

Don't force these records of the names used to do double duty as the labels that make the spreadsheet intelligible to humans. Separate the roles. One way to do this, but not the only one, is to

- put the labels down the left side, or occasionally along the top, of the values being identified
- record the names down the right side of the spreadsheet, or occasionally below, the areas that have been named
- give the names a distinctive formatting, perhaps italics.

Use Formulas | Defined Names | Create from selection to add to the workbook names in bulk, in a way that ensures that they match what is documented.

Choose names to that related items sort next to each other in an alphabetic listing.

### 2.12 Names tutorial I: Row and column matching

**ROW MATCHING IS POSSIBLE AS WELL AS COLUMN MATCHING**

In Lesson F we saw that when confronted by a formula that mentions several cells where one would be more usual, Excel can often make sense of the expression. It will resort to column matching to pick out one of the cells to act on.
The process Excel goes through to work out which cell to pick from a range doesn’t only involve column matching. It can involve row matching too.

**EXAMPLE OF COLUMN MATCHING**

We have already seen column matching in Lesson F.

**EXAMPLE OF ROW MATCHING**

1. Create a new workbook
2. In cell C1, write “Radius”
3. Type some numbers into cells C2:C11.
4. Highlight cells C1:C11
5. Use Excel’s Formulas | Defined Names | Create from selection command with the “Top row” command selected
6. Press F5, the shortcut for Goto, and satisfy yourself that a name “Radius” has been created. Dismiss the Goto dialog when you have done so.

7. In cell E1, write “Area”
8. In cell E2, write =PI() * Radius^2
10. See that Excel is using row matching to pick out the radius of each circle listed to calculate its area.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>Radius</td>
<td></td>
<td>Area</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td></td>
<td>=PI() * Radius^2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>2</td>
<td></td>
<td>=PI() * Radius^2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>2</td>
<td></td>
<td>=PI() * Radius^2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td>2</td>
<td></td>
<td>=PI() * Radius^2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>11</td>
<td>2</td>
<td></td>
<td>=PI() * Radius^2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>13</td>
<td>2</td>
<td></td>
<td>=PI() * Radius^2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>17</td>
<td>2</td>
<td></td>
<td>=PI() * Radius^2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>19</td>
<td>2</td>
<td></td>
<td>=PI() * Radius^2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>23</td>
<td>2</td>
<td></td>
<td>=PI() * Radius^2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>29</td>
<td>2</td>
<td></td>
<td>=PI() * Radius^2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**EXAMPLE OF ROW AND COLUMN MATCHING SIMULTANEOUSLY**

1. Create a new workbook
2 Press Ctrl+A. This selects all the cells on Sheet1.

3 Using Excel's Formulas | Defined names | Define name command, define a name "ValueOnSheet1" so that it refers to all the cells just selected.

4 Put some random entries in cells in the first few rows on Sheet1.

5 Move to Sheet2. Type =ValueOnSheet1 into a cell and copy it into the first few rows of that sheet.

6 See that where there you have put entries in Sheet1, they appear in corresponding cells on Sheet2. To do this Excel is matching both the rows and the columns in order to reduce a two dimensional range of cells to a single value that it can usefully place in a cell.

2.13 Names tutorial J: Aggregation

In Lesson F we saw that when confronted by a formula that mentions several cells where one would be more usual, Excel can often make sense of the expression. It will resort to matching columns to pick out one of the cells to act on. In Lesson I we saw that, if the spreadsheet layout demands it, Excel can match rows too.

In the first example below, a name Costs has been defined to refer to the shaded cells. The other populated cells all say =Costs. That formula refers to five cells, but Excel can only fit one value in each cell, so it chooses one by matching columns. In columns A and G, no column match is possible that intersects with the five yellow cells, so an error is displayed.

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>#/VALUE!</td>
<td>100</td>
<td>105</td>
<td>110.25</td>
<td>115.7625</td>
<td>121.5506</td>
<td>#/VALUE!</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>#/VALUE!</td>
<td></td>
<td></td>
<td>110.25</td>
<td>115.7625</td>
<td>121.5506</td>
<td>#/VALUE!</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>#/VALUE!</td>
<td>100</td>
<td>105</td>
<td>110.25</td>
<td>115.7625</td>
<td>121.5506</td>
<td>#/VALUE!</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>#/VALUE!</td>
<td></td>
<td></td>
<td>110.25</td>
<td>115.7625</td>
<td>121.5506</td>
<td>#/VALUE!</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>#/VALUE!</td>
<td>100</td>
<td>105</td>
<td>110.25</td>
<td>115.7625</td>
<td>121.5506</td>
<td>#/VALUE!</td>
<td></td>
</tr>
</tbody>
</table>

In the second example below, the formula in the populated cells says =SUM(Costs). Though there are five cells in Costs, the action of the SUM is to distill them into a single value, the total of the five cells. Now that that there is just one value, it can be fitted in any cell, and no column match is necessary.

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>552.5631</td>
<td>552.5631</td>
<td>552.5631</td>
<td>552.5631</td>
<td>552.5631</td>
<td>552.5631</td>
<td>552.5631</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>552.5631</td>
<td>100</td>
<td>105</td>
<td>110.25</td>
<td>115.7625</td>
<td>121.5506</td>
<td>552.5631</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>552.5631</td>
<td>552.5631</td>
<td>552.5631</td>
<td>552.5631</td>
<td>552.5631</td>
<td>552.5631</td>
<td>552.5631</td>
<td></td>
</tr>
</tbody>
</table>
SUM is not the only function that the values in the cells it acts on into a single value. Others are PRODUCT, AVERAGE, VAR, STDEV, MIN, MAX, INDEX, NPV and IRR. No column or row matching happens with these functions because the aggregating action renders it unnecessary.

2.14 Names tutorial K: Controlling aggregation

In earlier lessons, we have seen that Excel will resort to matching rows and columns where necessary to make sense of formulas that involve names, but that behaviour is unnecessary when dealing with functions that aggregate their inputs.

Most of the time that is appropriate behaviour, but there are occasions when it is not.

In the example below, a spreadsheet is seeking to partition net earnings between a positive part, labelled profit, and a negative part, labelled loss.

1 Define a name Earnings to refer to cells B2:F2. You can do this,
   • either, by selecting cells B2:F2 and using Excel's Formulas | Defined names | Define name command
   • or, by selecting cells A2:F2 and using Excel's Formulas | Defined Names | Create from selection command with the Left column option activated.

2 Select any one cell.

3 Use Excel's Formulas | Defined names | Defined name | Apply Names command to apply the just-made name Earnings.

4 See that Excel has changed the formula in line 6 to =MAX(0,-Earnings), but it has made no change to the formula in line 4.

We learned about Excel's Apply names command in Lesson E, where we saw:

TIP: The golden rule is that Apply names never makes a substitution that would cause the meaning of a formula to change. Every time that you expect Excel to substitute a name for some coordinates in your formula, and it doesn't, is an opportunity to revisit your understanding of what the formula would have meant had the substitution happened, and why it is not the same as what you have written in coordinate form.

So why does ApplyNames decline to change =MAX(0,F3) to =MAX(0,Earnings)? The reason is that MAX(0,F3) is finding the higher of two numbers. MAX(0,Earnings) would
find the higher of six numbers, the zero and the five values in the range now named Earnings. Excel does not adjust the formula because it detects that to do so would be to change the meaning of the formula.

MAX(0,Earnings) would be equivalent to MAX(0,F3) if Excel performed a column match on Earnings. But as we learned in the last lesson, MAX is one of the functions that aggregates its inputs, and renders column matching unnecessary.

Why, then, did Excel make a name substitution in the other formula? What was =MAX(0,-F3) has become =MAX(0,-Earnings). Excel would only change the formula if it was safe to do so. And it is safe to do so: while MAX is an aggregating function, which causes Excel not to column match, the minus sign, or any other mathematical operation, has the effect of re-enabling it.

This being so, it’s easy to get Excel to use names in the Profit formula also. Simply change =MAX(0,F3) to =MAX(0,+F3). Once that change is made, Excel will happily apply the name Earnings, so that the formula becomes =MAX(0,+Earnings).

Remember

- =MAX(0,Earnings) finds the largest of six numbers, the zero and the five numbers named Earnings
- =MAX(0,+Earnings) finds the largest of two numbers, the zero and whichever of the Earnings numbers is selected by column matching.

2.15 Names tutorial L: Naming constants

When will we be 10,000 days into the millenium that started on 1 Jan 2000? Excel can easily calculate that, from the formula =$MilleniumEnd+20000. Excel might interpret this formula by

- looking up the definition of MilleniumEnd, which is (say) =Sheet1!$A$1
- looking inside cell Sheet1!$A$1, and finding 31 Dec 1999 stored as a datecode
- adding 10,000 days
- displaying the answer, which should be formatted as a date

The process of evaluating MilleniumEnd is a two stage process, in that Excel has to look up the definition to see that is Sheet1!$A$1, and then look in that cell to dig out a value.

Excel permits this process to be reduced to a one stage process. Rather than defining MilleniumEnd as cell A1, and storing 31 Dec 1999 in that cell, one can associated MilleniumEnd with 31 Dec 1999 directly. Here is how.

1 Invoke Excel’s Formulas | Defined names | Define name command.
2 The upper field, labelled Names in workbook, will be empty, or may have some piece of text in it that Excel has found near to the cells that happen to be selected, and offered as
a possible candidate for the name to be defined. Either way, type into that field "MilleniumEnd"

3 Press Tab a couple of times to get to the lower field, labelled Refers to. That will have been prepopulated by Excel with the address of the cells that happen to be currently selected, since most times this command is used, it is the current selection that is to be named. But here, we don't want to name that selection; we want to name 31 Dec 1999. Replace the cell address by typing in that date.

4 Press OK to dismiss the dialog box.

Now Excel can interpret =MilleniumEnd+20000 more directly, by

· looking up the definition of MilleniumEnd, which this time it finds is 31 Dec 1999 without any need to look in any cells
· adding 10,000 days
· displaying the answer, which should be formatted as a date.

Naming a constant in this way allows assumptions to be provided to a spreadsheet without providing any cells for it. That can be a good thing; it means that the spreadsheet is not cluttered with assumptions any more than necessary, and are out of harm’s way in terms of unwise alteration by unskilled users of the spreadsheet. But it can be a bad thing; the spreadsheet has assumptions that are not obvious. It is best confined to constants that really won't ever change, like the date of the end of the millenium or the speed of light.

A good use is in error messages. It's quite common to write =IF(ABS(A1)>0.0001,B2/A1,“N/A”); but that might be better written as =IF(ABS(A1)>0.0001,B2/A1,MessageNa). Then MessageNa can be defined as ="N/A". At a later time, that name can be redefined to some other text string, which might be more appropriate in another language. If there are thousands of formulas like that, scattered over several worksheets, making the equivalent change manually would be painful.

2.16 Names tutorial M: Fine print

There are many small details that we left out of this tutorial, in order to avoid distracting from the main purpose, which is to explain how to use Excel's naming features as concisely as possible.

Here are some of those details.

LESSON A: GOTO AND LESSON B: DEFINING NAMES

Goto (F5) is not the only way to Goto a particular cell and Formulas | Defined names | Define name is not the only way to define a name. Microsoft's literature illustrates another method, which is to use the area next to the formula bar. Microsoft calls this space the name box.
The Name box acts as a combination of the Goto and the Define names commands.

- Type a cell address into it, and Excel will make that the selection, just like the Goto command.

- Select an area, and then type a name into the name box, and that name will be associated with the area, just as it would have been through the Formulas | Defined names | Define name command.

- Click on the arrow to the right of the name box, and a list will drop down, filled with names that have been defined in the workbook or worksheet. Choose one, and you will be taken to the area associated with the name, just as you would have been if you had pressed F5 and selected the name from the Goto dialog.

- Rather than selecting a name from the list, you can type it into the Name box and hit Enter, and Excel will select the relevant area. (But be careful. If you mistype the name, Excel will add a new name, defining the mistyped name as referring to the current selection.)

We find this last point in brackets very unattractive

- in principle: Is Excel reading from the list of names, or writing to it? The decision, is context sensitive, depending on what you type, and what has been typed in the past.

- in practice: Excel permits you to mess up the list of carefully defined names, without warning you about it.

There are several other reasons why we don't recommend using the name box.

- There is no way to reach the name box from the keyboard. It is activated by clicking in it, which involves removing one’s hands from the keyboard. While not everyone favors the keyboard over the mouse, we don't like actions that don't offer the option.

- In admittedly very old versions of Excel, the drop down box didn't exist at all.

- In some versions of Excel, the name box drop-down list only shows the first few dozen of the names that have been defined. If you want to interact with a larger number of names, you are going to have to use the Goto and Formulas | Defined names | Define name commands anyway. You might as well get into the habit from the outset and use those basic commands all the time.

- In versions of Excel before 2007, the width of the Names box is fixed, and not very large. Longer names are truncated. Similar names are indistinguishable. Excel 2007 fixes this by making the Names box resizable.
We favor the Goto and Formulas | Defined names | Define name commands as they have been present in a consistent way in every version of Excel from the first, and don't suffer any of these unattractive qualities.

LESSON B: DEFINING NAMES

In our example, the cells that we wanted to name happened to have labels nearby that said Sales and Costs. Excel offered them as suggested candidates for names when we used the Insert Name Define Command.

Where a label consists of more than one word, as in Capital expenditure, Excel would propose Capital_expenditure as the suggested name. The reason is that names can be made up of letters, digits, the underscore, and a full stop (period). Other characters are not allowed, so Excel replaces the space, which is not allowed, with an underscore, which looks similar and is allowed.

Operis’s own style is not to accept Excel's suggestions when they involve underscores. Underscores are hard to see next to the spreadsheet's grid lines, and can be mistaken for spaces or, less commonly, minus signs.

They also subconsciously disrupt the flow of the formula when reading it. =Capital_Expnditute+Income_Tax reads like Expenditure+Income embedded in the middle of some other material.

Instead, Operis recommends using a mixture of upper and lower case letters to indicate the word boundaries, as in CapitalExpenditure-IncomeTax. Some people call this mixture of letters camel case, or bumpy case.

LESSON D: NAMES ADJUSTMENT

The way name definitions are adjusted automatically when the cells they refer to change location is consistent with the way coordinate references in a formula are adjusted when the cells they refer to move.

However, the effect of moving a formula expressed in names is frequently not the same as the effect of moving a formula expressed in coordinates.

- Cut =$C$7 from C10 and paste it into D10: It will still refer to $C$7.
- Cut =Costs from C10 and paste it into D10: Assuming Costs is defined as =$C$7:$M$7, the formula will change from referring to C7 to referring to D7.

The formula is unchanged, but its interpretation in a new location will be different due to column matching.

Some see this difference in behaviour between formulas expressed in names and formulas expressed in coordinates as confusing. Others consider it valuable; formulas
expressed in names will always refer consistently to columns, no matter how much mutilation they suffer with cutting and pasting.

LESSON E: APPLY NAMES

The Apply names command obeys the One Cell Rule. That's why we included the direction in Step 2 "Select the cell which calculates profit using coordinate references...". It's partly because that cells contains a formula, and we want to see how it is affected when we start using the Apply names command. But it's also so that the relevant cell will in fact be acted upon by the command.

The Apply names command described in this lesson is built into Excel. OAK has a command, also called Apply Names, which looks almost identical and performs the same action, but is more convenient and more powerful.

LESSON I: ROW AND COLUMN MATCHING

A non-obvious point is that row and column matching is connected with names only indirectly. Expressions such as =Costs are resolved to =D7:M7 (or whatever the name definition indicates). That formula cites several cells and will typically provoke column matching.

The column matching would equally happen if the formula had been written in coordinate notation. To this extent, it is not a property of the names directly, but a fundamental mechanism of the way Excel interprets a formula that is indirectly activated when names are used.

LESSON K: CONTROLLING AGGREGATION

The Apply names command obeys the One Cell Rule. That's why we included the direction in Step 2 "Select any one cell."

Opinion is divided on the technique taught here. Operis finds it valuable, uses it and teaches it. Others feel that it demands a finely detailed knowledge of how Excel works that is not universal, and therefore not suitable for use in spreadsheets that are to be widely accessible. Common ground can be found in the advice that wherever there is ambiguity, the clearest possible formulation should be used.

2.17 Names tutorial N: Summary

Opinion is more divided on whether it's a good idea for formulas to identify their inputs by names rather than coordinates in formulas than by any other topic in the field of developing spreadsheet models.
ARGUMENTS FOR USING NAMES

Names make a formula clearer and easier to read. The intent of =Revenue-Costs is clearer than =P999-P807.

This is particularly so when formula refer to cells on other workbooks. The name formula would still be =Revenue-Costs; the coordinate formula would be ='Profit and loss'!P999-'Profit and loss'!P807. In longer formula, the clutter of worksheet names is particularly impenetrable.

A naming convention that is agreed by a team and used consistently within it greatly aids different team members from picking up and understanding each others' models.

Many methodologies for developing spreadsheets focus on laying out the spreadsheet so that related items are physically near to each other. In some cases, this involves extensive repetition, to stage values for input to and output from sub-parts of a model. Referring to quantities using meaningful names allows one to worry less about where quantities happen to lie physically on a spreadsheet. What matters is whether the terms are semantically close, and whether related quantities can be found near each other in the alphabetic listings that Excel offers with the various commands for manipulating names.

Names facilitate code reuse. Model A may contain a complex tax calculation in an obscure jurisdiction. Model B needs to be fitted with the same calculation. If both models are expressed in coordinates, there is no chance that the fragment of model A that contains the calculation will have references that are consistent with where the relevant inputs fall in model B. If the two models have been developed using names, and those names follow a convention agreed within the team, then Model A's calculation may be copied into Model B and made to work quickly with little modification, without any concern for physical layout.

Relying on Excel to match rows and columns relieves the analyst of the need to do the same and eliminates completely a common cause of error, making reference to a cell which is in a different column from the one intended. It also forces a model developer to pursue consistent conventions about what columns are for in different parts, or on different worksheets, of a model.

ARGUMENTS AGAINST USING NAMES

Names are not widely understood by ordinary spreadsheet users. Using them makes a spreadsheet less accessible, and less safely modifiable, by such users.

The formula =P999-P807 refers unambiguously to the cells mentioned. Determining quite which cells the formula =Revenue-Costs refers to needs some work, firstly, to check that those names are associated with the intended cells, and secondly to fathom out the effect of any row or column matching that may apply.

What happens when a formula expressed using names is cut or copied to a new location is not the same as what happens when a formula expressed in coordinate notation is so
moved. The latter is widely understood; the former much less so, and is a source of error in the hands of recipients of a spreadsheet untrained in the behavior of Excel names.

This last point is particularly an issue when a formula that uses a name is cut or copied from one workbook to another. Excel will make new names in the destination workbook in a process over which the user has no control and may not be aware of.

Two spreadsheets can have identical formulas but act differently because the names are differently defined. This kind of inconsistency can be hard to detect.

Using names reduces the opportunity to transfer a model from Excel into other spreadsheet packages. Not all of them can make sense of names, though increasingly many can.

INDEPENDENT RESEARCH

The one bit of published research on the use of names was presented to the Eusprig conference in Paris in 2009. Ruth McKeever, Kevin McDaid & Brian Bishop of Dundalk Institute of Technology presented Analysis of the Impact of Named Ranges on the Debugging Performance of Novice Users. It found that a group of novice spreadsheet users found a lower proportion of the errors deliberately seeded into a spreadsheet that made use of names than did a similar group presented with errors and spreadsheets that were identical except that they used coordinate references throughout and no names.

Areas that would be interesting for future research are

- whether the same detection rates would apply in groups that had been made fluent in the use of names before the exercise started
- the rate of faults over the whole life cycle of a spreadsheet, ie during the development process as well as a review stage, rather than a review stage alone.
Basic concepts in OAK
3.1 Basic concepts in OAK

This area of the help covers the ideas that underpin a number of the OAK commands, particularly

- expression simplification
- expression pruning
- discrepancy analysis
- reconstructing calculations
- precedent substitution
- removing names from spreadsheets that use them.

3.2 Expression Simplification

The formula

\[
=\text{IF}(A1+B2+C3+D4>E6+F7+G8+H9, A1+B2+C3+D4,E6+F7+G8+H9)
\]

can be expressed more simply as

\[
=\text{MAX}(A1+B2+C3+D4,E6+F7+G8+H9)
\]

The second formula is better than the first as it

- is shorter, and so easier to read
- mentions the two subexpressions once each rather than twice, so it is more robust to change.

OAK is able to transform long expressions of this kind into shorter ones automatically. It does so by recognising patterns in formulas, and replacing them by equivalents according to a number of algebraic rules. In the example above, the rule is

\[
=\text{IF}(x>y,x,y) \quad \text{maps to} \quad \text{MAX}(x,y)
\]

OAK recognises that the first formula matches the pattern, with \(x=A1+B2+C3+D4\), and \(y=E6+F7+G8+H9\). It then replaces the \(\text{IF}(...)\) in the formula with \(\text{MAX}(x,y)\), substituting the terms it has matched to \(x\) and \(y\) as necessary.

OAK is equipped with about fifty algebraic rules which it can apply to simplify formulas. They are activated by the Formula | Optimize command.

3.3 What's safe in simplification

SAFE SIMPLIFICATION

Some of the actions that OAK can take to simplify formulas can be performed completely safely; that is, the simplified formula will always give the same answer as the original
formula. For example, =A1++B1 can be rewritten as =A1+B1, and depended on to give the same answer.

The rule that OAK follows to make transformation just shown is that \( x++y \) maps to \( x+y \). This rule is always safe, by which we mean that the transformation it describes can be made without there being the slightest risk that the meaning of the formula, or the results it generates, will be altered.

This is true whatever \( x \) and \( y \) are. They can be

- cell references, as in the \( =A1++B1 \) example with which we have started
- ranges, as in \( =A5:A9++B5:A9 \)
- ExcelNames, as in \( =Revenue++Costs \)

OAK terms the transformation from \( x++y \) to \( x+y \) as conservative, in that it is safe under any conditions.

**UNSAFE SIMPLIFICATION 1**

The example with which we introduced expression simplification had the pattern \( =IF(x>y,x,y) \) mapping to \( MAX(x,y) \). That simplification is fine so long as \( x \) and \( y \) are coordinate references.

But if \( x \) and \( y \) are name references, there's a good chance that the two formulas will give different answers. \( =IF(TaxDue>Threshold, TaxDue, Threshold) \) is not at all the same as \( MAX(TaxDue,Threshold) \). The reason has to do with Excel's propensity to pick out single cells from ranges by matching rows and columns.

Similarly, the expression \( =AVERAGE(B5+0,B8*1) \) is equivalent to \( =AVERAGE(B5,B8) \). But it does not follow from that that we can substitute \( x \) whenever we encounter \( x+0 \), or \( y \) whenever we encounter \( y*1 \). \( =AVERAGE(B5:D5+0,B8:D8+0) \) is not at all the same as \( AVERAGE(B5:D5,B8:D8) \). In the first, Excel will use column matching to pick out two cells to average; in the second, it will calculate the average of six cells. The rules Excel follows in such circumstances are in Lessons I, J and K of the Excel names tutorial in this help.

**UNSAFE SIMPLIFICATION 2**

The expression \( =IF(A1>B2,C3,0) \) can be rewritten as \( =(A1>B2)*C3 \). Not everyone would think that the second formulation was any clearer, but some people like it.

But it is not safe to infer a rule that \( =IF(x,y,0) \) can map to \( =x*y \). Such a rule would work splendidly if \( x \) evaluates to TRUE or FALSE, since those truth values are coerced by Excel into the integers 1 and 0 the moment they are involved in any maths.

But what happens \( x \) is not a truth value? Suppose instead it was an expression that evaluated to 4. Just as Excel will turn truth values into 1 and 0 integers when the context
demands it, so it will turn numbers into truth values. Since Excel expects the first parameter of the IF() function to be a truth value, it will coerce the 4 into one. The convention is that 0 is treated as FALSE, and any other value is taken as TRUE. IF(4,y,0) therefore evaluates to IF(TRUE,y,0), which in turn is \( y \).

If we apply our rule, we will map this expression from \( =IF(x,y,0) \) to \( x*y \), which will come out as \( 4*y \), which is not the same answer as the untransformed expression.

OAK terms the transformations shown here as aggressive. They will work in the majority of real world conditions, but it can't be relied upon to be safe whatever \( x \) is.

### 3.4 Simplification rules

OAK is able to transform long expressions into shorter ones automatically. It does so by recognising patterns in formulas, and replacing them by equivalents according to a number of algebraic rules.

The rules are divided into ones that are conservative, that is, always safe to apply, and others that are more aggressive: they are fair substitutions in most circumstances, but can alter the meaning of a formula on other occasions.

#### CONSERVATIVE RULES

<table>
<thead>
<tr>
<th>Original expression</th>
<th>Replacement</th>
</tr>
</thead>
<tbody>
<tr>
<td>(-x+y)</td>
<td>( y-x )</td>
</tr>
<tr>
<td>(x-y^*x)</td>
<td>( x^*(1-y))</td>
</tr>
<tr>
<td>(y^*x-x)</td>
<td>( x^*(y-1))</td>
</tr>
<tr>
<td>(a^*x-b^*x)</td>
<td>((a-b)^*x)</td>
</tr>
<tr>
<td>(a^*x+b^*x)</td>
<td>((a+b)^*x)</td>
</tr>
<tr>
<td>IF((x&gt;y,x,y))</td>
<td>MAX((x,y))</td>
</tr>
<tr>
<td>IF((x&gt;=y,x,y))</td>
<td>MAX((x,y))</td>
</tr>
<tr>
<td>IF((x&lt;y,y,x))</td>
<td>MAX((x,y))</td>
</tr>
<tr>
<td>IF((x&lt;=y,y,x))</td>
<td>MAX((x,y))</td>
</tr>
<tr>
<td>IF((x&lt;y,x,y))</td>
<td>MIN((x,y))</td>
</tr>
<tr>
<td>IF((x&gt;=y,x,y))</td>
<td>MIN((x,y))</td>
</tr>
<tr>
<td>IF((x&gt;y,y,x))</td>
<td>MIN((x,y))</td>
</tr>
<tr>
<td>IF((x&lt;=y,y,x))</td>
<td>MIN((x,y))</td>
</tr>
<tr>
<td>(x++y)</td>
<td>(x+y)</td>
</tr>
<tr>
<td>(x--y)</td>
<td>(x+y)</td>
</tr>
<tr>
<td>(x-+y)</td>
<td>(x-y)</td>
</tr>
</tbody>
</table>
\begin{align*}
x+y & \quad \rightarrow \quad x \cdot y \\
(x+y)z & \quad \rightarrow \quad x+y \cdot z \\
x \cdot (y+z) & \quad \rightarrow \quad x \cdot y \cdot z \\
x \cdot (y-z) & \quad \rightarrow \quad x \cdot y + z \\
x - (y+z) & \quad \rightarrow \quad x \cdot y + z \\
-x-y & \quad \rightarrow \quad y \cdot x \\
-x+y & \quad \rightarrow \quad -x \cdot y \\
x \cdot 0 & \quad \rightarrow \quad 0 \\
0 \cdot x & \quad \rightarrow \quad 0 \\
(x \cdot y)z & \quad \rightarrow \quad x \cdot y \cdot z \\
x \cdot (y^{*}z) & \quad \rightarrow \quad x \cdot y^{*}z \\
x \cdot (y+z) & \quad \rightarrow \quad x \cdot y + z \\
x \cdot (y-z) & \quad \rightarrow \quad x \cdot y - z \\
(x+y)+z & \quad \rightarrow \quad x+y \cdot z \\
(x+y)+z & \quad \rightarrow \quad x \cdot y + z \\
x+(y-z) & \quad \rightarrow \quad x+y - z \\
(x-y)+z & \quad \rightarrow \quad x-y + z \\
\text{AND}(\text{NOT}(x),\text{NOT}(y)) & \quad \rightarrow \quad \text{NOT}(\text{OR}(x,y)) \\
\text{OR}(\text{NOT}(x),\text{NOT}(y)) & \quad \rightarrow \quad \text{NOT}(\text{AND}(x,y)) \\
\text{IF}(x,\text{TRUE},\text{FALSE}) & \quad \rightarrow \quad (x) \\
\text{IF}(x,1,0) & \quad \rightarrow \quad (x) \\
\text{IF}(x,y,0) & \quad \rightarrow \quad ((x) \cdot (y)) \\
\text{IF}(x,\text{FALSE},\text{TRUE}) & \quad \rightarrow \quad \text{NOT}(x) \\
\text{IF}(x,y,y-z) & \quad \rightarrow \quad (y \cdot \text{IF}(x,0,z)) \\
\text{IF}(x,y-z,y) & \quad \rightarrow \quad (y \cdot \text{IF}(x,z,0)) \\
((x)) & \quad \rightarrow \quad (x) \\
\end{align*}

\textbf{AGGRESSIVE RULES}

\begin{tabular}{|l|l|}
\hline
Original expression & Replacement \\
\hline
+x & x \\
-x & x \\
x\cdot 1 & x \\
1 \cdot x & x \\
\hline
\end{tabular}
x+0      x
0+x      x
x-0      x
0-x      -x
IF(x,-y,-z) -IF(x,y,z)
-IF(x,0,-y) IF(x,0,y)
-IF(x,-y,0) IF(x,y,0)
-IF(x,-y,-z) IF(x,y,z)
x=TRUE     x
x<>FALSE     x
IF(x,y,y+z) (y+IF(x,0,z))
IF(x,y+z,y) (y+IF(x,z,0))
IF(x,y-y-z) (y-IF(x,0,z))
IF(x,y-z,y) (y-IF(x,z,0))
IF(w,x*z,y*z) (IF(w,x,y)*z)
IF(w,z*x,z*y) (IF(w,x,y)*z)
IF(w,x/z,y/z) (IF(w,x,y)/z)

3.5 Expression Pruning

What is the difference between these two formulas?
• =INDEX(B15:Z19,4,7)*INDEX(B25:Z29,2,4)
• =H18*E26

The difference between them is at the same time
• considerable, in that the second is much shorter and easier to understand
• negligible, in that both formulas multiply the same cells together.

OAK provides facilities for taking formulas that use complicated conditional functions, such as CHOOSE, INDEX, VLOOKUP and OFFSET, and changing them so that they show the cells that are really involved.

Operis calls this action pruning because it decodes the formula and makes an internal representation that resembles a tree.

For example, =IF(A1>B2,C3+D4,E5*F6) is represented as
Assuming that the value in cell A1 is larger than the value in cell B2, then the result of the IF will be C3+D4, the branch of the tree marked "then". In such circumstances, OAK trims away the "test" and the "else" branches of the IF, and preserves simply the "then" branch, leaving a formula that is much easier to read.

This expression pruning is offered by OAK in three contexts.

- OAK's Formula | Prune Inactive Path command will make transformations directly to the formula in the currently selected cells.
- OAK's technologies for preparing reconstructions of formulas can generate long expressions as the hierarchy of precedent cells fans out. This fan-out can be controlled by discarding the branches that are not active at any moment. (It is for this purpose that the expression pruning was developed.)
- Similarly, OAK's reports that explain the discrepancies between two cells can become very large due to precedent fan out, and the expression pruning can control this.

### 3.6 Pruning rules

OAK's expression pruning feature offers control over the expressions it can remove from formulas.
Being able to activate the functions separately means that a disagreeable formula such as

$$=\text{IF(INDIRECT(...),OFFSET(...),VLOOKUP(...))}$$

can be pruned in stages, so that it's possible to see what the awkward terms actually evaluate to and what underlying calculation the IF is performing.

The rules OAK follows when prunes these functions are:

BEFORE PRUNING

AVERAGEIF(conditionrange, testvalue, valuerange)

AVERAGEIFS(valuerange, criteriarange1, criteria1, criteriarange2, criteria2, ...)

CHOOSE(K, item1, item2, ..., itemK)

AFTER PRUNING

AVERAGE(subrange)

AVERAGE(subrange)

itemK

COMMENTS

subrange is a multi-area range consisting of those cells in valuerange in which the equivalent cells in conditionrange satisfy testvalue

subrange is a multi-area range consisting of those cells in valuerange for which the corresponding cells in criteriarangeX satisfy criteriaX
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COLUMN(range)</td>
<td>the column value of the top left cell in the specified range, or in the case where no range is specified, the column where the formula is located.</td>
</tr>
<tr>
<td>COUNTIF(range,testvalue)</td>
<td>COUNT(subrange)  subrange is a multi-area range consisting of those cells in range which satisfy test value.</td>
</tr>
<tr>
<td>COUNTIFS(criteriarange1, criteria1, criteriarange2, criteria2,...)</td>
<td>COUNTA(subrange)...  subrange is a multi-area range consisting of those cells in the union of criteriarangeX satisfying criteriaX.</td>
</tr>
<tr>
<td>HLOOKUP</td>
<td>a reference identifying the cell that containing the value that would be returned by the HLOOKUP given its current inputs See note 2</td>
</tr>
<tr>
<td>IF(test,truepart,falsepart)</td>
<td>truepart OR falsepart Depending on whether test is TRUE or FALSE</td>
</tr>
<tr>
<td>INDEX(range,x,y)</td>
<td>subrange  subrange is a portion of range, typically a single cell, but possibly a range one row high or one column wide (if x or y are zero)</td>
</tr>
<tr>
<td>INDIRECT(...)</td>
<td>a reference identifying the range that would be returned by the function given its current inputs</td>
</tr>
<tr>
<td>LOOKUP(...)</td>
<td>a reference identifying the range that would be returned by the function given its current inputs</td>
</tr>
<tr>
<td>MATCH(value, array, matchtype)</td>
<td>the value in the array that matches the specified value according to the matchtype</td>
</tr>
</tbody>
</table>
MAX(range)  MAX(cell)  MAX is retained to indicate what was there before, but has no effect as it is given just one parameter.

MIN(range)  MIN(cell)  MIN is retained to indicate what was there before, but has no effect as it is given just one parameter.

OFFSET(...)  a reference identifying the range that would be returned by the function given its current inputs

ROW(range) or ROW()  the row value of the top left cell in the specified range, or in the case where no range is specified, the row where the formula is located.

SUMIF(conditionrange,testvalu e,valuerange)  SUM(subrange)  subrange is a multi-area range consisting of those cells in valuerange in which the equivalent cells in conditionrange satisfy testvalue

SUMIFS(valuerange, criteriarange1, criteria1, ...criteriaRange2, criteria2...)  SUM(subrange)  subrange is a multi-area range consisting of those cells in valuerange for which the corresponding cells in criteriarangeX satisfy criteriaX

VLOOKUP(...)  a reference identifying the cell that containing the value that would be returned by the VLOOKUP given its current inputs  See note 2

NOTES

1 Expressions are always relative after pruning, regardless of what combination of relative and absolute formulas went to make up the original expression.
2. The wording used to describe the results of pruning HLOOKUP and VLOOKUP is intentionally different from the wording used to describe the results of pruning INDEX, INDIRECT, and LOOKUP. This is because INDEX, INDIRECT, and LOOKUP return references, which are dereferenced or not as appropriate in the context. For example, =ROW(INDEX(...)) returns a useful result, because Microsoft has engineered Excel so that INDEX returns a reference; it does not turn that reference into a value until required to evaluate an expression. But HLOOKUP and VLOOKUP return values, not references; Excel will not let you even enter a formula such as =ROW(HLOOKUP(...)). Nevertheless, OAK does compute the reference that would be returned by HLOOKUP and VLOOKUP if they behaved like INDEX, INDIRECT, and LOOKUP and it is that reference that is put in a pruned expression.

3.7 Difference between simplifying and pruning formulas

OAK offers two mechanisms that can alter a formula. It can simplify them, and it can prune them. The two are similar in that they make the formula shorter and easier to read. But there are important differences between them and it is important not to confuse them.

SIMPLIFICATION

OAK can simplify a formula by matching them against patterns specified in rules describing algebraic transformations.

The formula =IF(A1>B1,A1,B1) can be rewritten as =MAX(A1,B1). OAK has a set of rules that it can use to alter expressions through algebraic transformations.

The formula will give results in its simplified form that are identical to the one it gave in its original form, no matter what inputs are provided to it....

...subject to fine print concerning transformation rules that are conservative or aggressive.

Offered by OAK
• in the Formula | Optimize command

PRUNING

OAK can prune a formula by evaluating conditional elements of the expression and discarding the bits of the expression that don't apply.

If the value in cell A1 is bigger than the one in cell B1, then the formula =IF(A1>B1,A1,B1) evaluates to the contents of B1. One could replace the IF by writing =A1.

The formula above would continue to give the same answer after it has been pruned as it did before, only so long as A1 contained to exceed B1.

(The notion of conservative or aggressive treatment is not relevant to pruning a formula)

Offered by OAK
• in the Formula | Prune Inactive Path command
SIMPLIFICATION

- as an option in Formula | Reconstruct
- as an option in the Formula | Analyze Discrepancies command

PRUNING

- as an option in the Formula | Reconstruct command
- as an option in the Formula | Analyze Discrepancies command

When developing a spreadsheet, you may like to keep the suggestions of OAK’s Formula Optimize command permanently. You will definitely not want to keep pruned formulas permanently. Pruning alters the spreadsheet, in a way intended to make it easier to understand, but which will lead the spreadsheet to give the wrong answer with different inputs. Make sure you keep a copy of the spreadsheet before pruning it.

3.8 Discrepancy Analysis

TRIANGULATING ON THE ANSWER

The single most effective technique for increasing the reliability of spreadsheets is to calculate key results more than once, using intentionally different methods. If the separate derivations give the same answers, one can have high confidence (but not certainty) that the results are dependable.

This approach can be taken within a spreadsheet...

- in a financial model that generates a balance sheet, it is a powerful check to derive the assets and liabilities separately, and to check that the resulting statement balances as it is meant to.

- the same is true for other financial presentations, such as a statement of sources and uses of funds. To calculate one of the elements as the balancing figure in the table is to squander a valuable opportunity for cross-checking one’s work.

...or it can be done between spreadsheets

- One way to check a spreadsheet is to build a second version of all or part of it and check whether the two calculations give the same result. Another way is to key the assumptions into a template model that is thought to be correct, and see whether the difference between the answers is small enough not to matter very much.

- Using either of these methods, it will frequently be the case that the two calculations do not give the same answer and the analyst will wish to work out why. Is the fault in the model being tested, or the model against which it is being compared? Or is there perhaps something wrong with both of them?
EXAMPLE

In the simple spreadsheet below, the analyst has attempted to follow this precept. He has derived totals for each row, and for each column, and then derived grand totals from each of them. But the yellow cells show that these row grand total and the column grand total don't match.

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Dogs</td>
<td></td>
<td>North</td>
<td>South</td>
<td>East</td>
<td>West</td>
<td></td>
<td>Total</td>
</tr>
<tr>
<td>3</td>
<td>Cats</td>
<td></td>
<td></td>
<td>7</td>
<td>12</td>
<td>5</td>
<td>1</td>
<td>25</td>
</tr>
<tr>
<td>4</td>
<td>Fish</td>
<td></td>
<td>14</td>
<td></td>
<td>20</td>
<td></td>
<td></td>
<td>34</td>
</tr>
<tr>
<td>5</td>
<td>Parrots</td>
<td></td>
<td>3</td>
<td>10</td>
<td>8</td>
<td></td>
<td>21</td>
<td>94</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Total</td>
<td>24</td>
<td></td>
<td>22</td>
<td>33</td>
<td>3</td>
<td></td>
<td>82</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Why don't the two totals match? OAK can help explain why not.

First, load the example spreadsheet. Instructions on loading example spreadsheets are found in the Example Spreadsheets section of this help.

Once the example spreadsheet is loaded into Excel:

1. Click on one of the yellow cells.
2. Hold the CTRL key down and click on the other of the yellow cells. (Holding the CTRL key down causes the second cell to be included in the selection without deselecting the first one, making a multi-area range.)
3. Now use OAK’s Analyze Discrepancies command.
4. OAK will present a dialog. Just accept the default settings for now, by clicking on OK or pressing Enter.

OAK will generate a new worksheet.
NARRATIVE

This analysis sets out

- near the top: the 94 and the 82, the values in the two cells that we selected before invoking the Analyse discrepancies command.
- between the 94 and the 82: the discrepancy between the two values. We want this value to be zero, but here it is -12.
- below the 94, the hierarchy of precedent formulas and values for that result, fanning out leftwards; in column B, are the four row totals; and below and to the left of each of those, in column A, are the values from the table that make up each of those.
- below the 82, the hierarchy of precedent formulas and values for that result, fanning out rightwards; in column F, are the four column totals; and below and to the right of each of those, in column G, are the values from the table that make up each of those.

**TIP:** These cells don't just happen to have the same values as the ones in the first spreadsheet. They contain simple formulas that mean they are linked to the originals cells. You can follow any of the links back by selecting the relevant cell and pressing CTRL+[ (left square bracket). Then when you have examined the source cell, you can press F5, then Enter, to return to the analysis spreadsheet.

The number at the left extremity of the analysis (column A) and at the right extremity (column G) are all drawn from the body of the table, but have been arrived at by different methods. OAK has matched the values on the left and on the right by adding distinguishing colors, and by linking the cells it has matched with hyperlinks.

**TIP:** The cells in these analyses have two kinds of links: hyperlinks, which can be followed by the mouse and which associate matching items on the left and right side of the discrepancy analysis, and formula links, which can be followed using CTRL+[ (left square bracket) and are linked back to the spreadsheet being analysed.

The second number down in the leftmost column, a 12, has no coloring and no hyperlinks, because OAK has been unable to match it to any item in the rightmost column. That is a good indication that it may be the cause of the mismatch between the two numbers being examined. The fact that it coincides exactly with the discrepancy at the top of the analysis reinforces this suspicion.

Select the cell containing the 12 and press Ctrl+[ to follow the link formula back to its source in the spreadsheet under test. Now press Ctrl+] (right square bracket) to see what dependents that cell has. You will see it has just one, in the row totals, but none in the column totals. Examination of the second row total, which displays 22, shows that it omits the first row from total. In fact, Excel hinted this all along, by warning with a green triangle in the upper corner of the cell that it was different from its neighbors.

### 3.9 Fan out

Imagine a spreadsheet in which every formula mentions four other cells. Something like =A1+B2+C3+D4, or =SUM(X5:Y6).
Then any cell that contains a formula has four precedents. And each of those precedents likely contains a formula that has a further four precedents. So the first cell has ...

- 4 direct precedents
- 16 precedents one level closer to the spreadsheet's inputs
- 64 precedents two levels away
- 256 precedents three levels away

...and so on. Eventually, the fan-out will stop, because the chain of precedents reaches the spreadsheet's inputs (unless the spreadsheet is circular). In real spreadsheets, this fan-out of precedents can be explosively large, and can be an obstacle to gaining any useful overview of how particular numbers are derived.

This fan out is evident in the physical layout of the example given of Discrepancy analysis. Taming it is a priority, as it undermines the usefulness of the discrepancy analysis output. OAK seeks to do this by making available as options in the discrepancy analysis.

- a limit to the number of levels of precedence that the discrepancy analyzer examines
- pruning of expressions: the discrepancy analyzer confines its attention to the branches of conditional expressions that are actually active, given the current spreadsheet inputs, and does not provide any analysis of non-active branches; this is the same mechanism as is offered by the OAK Review | Formula | Prune command
- the ability to specify a limit beyond which large ranges will not be followed.

Fan-out is also a consideration in the reconstructions of calculations that OAK can prepare, and the Formula | Reconstruct command offers similar facilities with the same purpose in mind.

### 3.10 Constraining discrepancy analysis

Discrepancy analyses can become large because cells’ precedent trees can fan out to a considerable degree. OAK offers a way to confine the development of a discrepancy analysis to just a few levels of precedent.
A large number for the maximum depth will produce a more substantial discrepancy analysis. A small number will cause the discrepancy analysis to be produced very much more quickly.

### 3.11 Expression pruning in discrepancy analysis

A formula of the kind

\[ =\text{VLOOKUP}(A1:Z1000,"London",3) \]

makes reference to 26,000 cells and every one of them would be listed if such a formula was encountered in a discrepancy analysis. And, depending on how many levels of precedent OAK had been instructed to report, each of those 26,000 cells would have precedents, possibly many of them, which OAK would also attempt to list.

The fan-out in the resulting report would be enormous. And quite useless. Though the VLOOKUP may mention thousands of cells, it will always evaluate to a single value corresponding to just one of them.

OAK has the capability of confining a discrepancy analysis to the one useful cell. It does this by subjecting formulas that it encounters to the same kind of expression pruning as is used in OAK’s Formula Optimize command. An analysis that may be thousands of cells without pruning can be reduced to a few dozen cells with it.

Which functions are to pruned are indicated half way down the discrepancy analysis dialog and can be specified by pressing the button to the right of the list.
Tip: If two calculations differ because one is taking the TRUE branch of an IF and the other is taking the FALSE branch, you will likely not want to prune away the IF function. But if they follow the same branch of the IF, but nevertheless deliver different results, pruning out the IF may well be helpful.

3.12 Large ranges

The expression SUM(A1:Z1000) is just 12 characters long, but it mentions 26,000 cells. Encountering such a term in a formula will cause an explosive fan-out in a discrepancy analysis.

OAK therefore provides the means to specify a limit beyond which formulas have terms that mention lots of cells are not further processed in a discrepancy analysis.

If this facility is enabled, and the default limit of 500 cells accepted, then the addresses mentioned in a range of up to 500 cells will be included in a discrepancy analysis (and, potentially, their precedents also); but ranges involving more cells will not be included in the analysis.
3.13 Associating references or values

A discrepancy analysis reports a fan-out of two cells, one working from the centre of the report leftwards, the other heading rightwards.

The purpose of the report is to show what is the same, and what is different, in the derivations of the two cells. To this end, OAK, identifies cells that coincide in the two precedent hierarchies.

For example, if cell 1 has the formula =A1+B2+C3, and cell 2 shows =B2+C3+Z9, then a discrepancy analysis will

- make a list of links to those precedent cells, in the order in which they are mentioned in the two formulas
- use color to signal that a reference to B2 appears among the precedents of both cells
- do the same thing for cell C3, with a different color to distinguish them from the first pair
- make it possible to navigate quickly between these pairs by hyperlinking them together
- leave the references to cells A1 and Z9 uncolored to show that they each appear in one of the formulas only.

The cells that are cross linked, through color and hyperlinks, will in many cases play corresponding roles in their respective calculations, but there is no guarantee that they will. They are merely the first references on either side of the analysis that OAK detected as coinciding.

For OAK to cross link pairs of cells in this way, the formulas must refer to exactly the same cell. However, there are times when OAK will give a more useful answer if it makes this pairing by value. Then cells will be indicated as matching if they contain the same thing, rather than concern the same address.

3.14 Suppressing unmatched subtrees

Imagine that a cell contains the formula =A1+B2+C3, and another shows =Z9+B2+C3. A discrepancy analysis will quickly shows that the cells are similar their second and third terms, but dissimilar in the first.

But if A1 in turn refers to three other cells, and those refer to three more in turn, the discrepancy analysis will be dominated by the precedents of A1 (the cells identified by a black border below).
These cells take up much space on the report, and cause the cells that are matched, as shown by color and hyperlinks, to be displaced relative to each other, making the result hard to read. This reduced readability comes at no advantage, in the sense that it exposes no cells that are common to both calculations.

OAK can be instructed not include subtrees such as these that turn out to have no contents that cross-match with cells on the other side of the report.

If this option is enabled, the display of the subtree is suppressed.
The subtree is still examined by OAK, and would be shown if any matching cells were detected. All that is suppressed is the inclusion of this part of the precedent hierarchy in the final report.

Tip: If a discrepancy analysis seems to contain much less information than you expect, check that this option is turned off.

### 3.15 Formula reconstruction

#### INTRODUCTION

Understanding which cells a formula refers to, and coming to a view on whether they are the correct cells in the circumstances, is a central activity in checking a spreadsheet.

Excel offers several helpful indications of these relationships between these cells, which are described in this help under Finding cell precedents in Basic Excel Concepts in this help.

#### THE PROBLEM

Imagine you have a formula \( =E3000+E6000+E9000 \).

Whatever cell that formula happens to be in, it's hard to see what quantifies it is adding up, because they are thousands of rows apart.

There are tricks, involving just a handful of keystrokes, which make use of the methods built into Excel for finding cell precedents, which can help decipher this formula. But the tricks will only work when all the cells are on the currently active worksheet. The formula \( =\text{Sheet1!E3000+Sheet2!E6000+Sheet3!E9000} \) is a fiddle to decipher using methods built in to Excel.

#### THE SOLUTION

This problem is so common that Operis has sought to address it by including relevant facilities in OAK. The most ambitious feature of OAK is the ability to generate a reconstruction of a formula. A reconstruction is a new worksheet which lays out the essence of a calculation as simply as possible.
FIRST EXAMPLE

The easiest way to understand a reconstruction is to try one.

**ACTION**

1. On a new worksheet, type in cell F9 the formula above, =E3000+E6000+E9000.

2. Put the value 10 in cell E3000, 20 in cell E6000, and 30 in cell E9000. The easiest way to do this is to press Ctrl+[ while cell C5 is selected, and visit the three cells mentioned in the formula in turn, filling them in and pressing TAB to move on to the next one.

3. Select cell F9 again. The easiest way to do this is the press F5, then enter.

4. Invoke the OAK Review | Formula | Reconstruct command. A dialog will appear, with lots of options. These will be explained later in this help. For now, accept the defaults, and press OK.

OAK will generate a new workbook, like the one below.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>I</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Item</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Original</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Reconstructed</td>
<td>60</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Discrepancy</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>WORKBOOK: Book1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>WORKSHEET: Sheet1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td>20</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Cell F2 in the new workbook has the formula =Book1!Sheet1!F9, which links it to the calculation that has been reconstructed.

Cells E10, E12 and E14 in the new workbook have the formulas =Book1!Sheet1!E3000, =Book1!Sheet1!E6000 and =Book1!Sheet1!E9000, which link to the cells that were referenced by the calculation that has been reconstructed.

Tip: The cell linked to by any of these four formulas can be seen in its original context by selecting the formula, then pressing Ctrl+[. You can return to the reconstruction report by pressing F5, then OK or Enter.
Cell F3 in the new workbook is OAK's reconstruction of the original formula. It is derived by taking the original formula, =E3000+E6000+E9000, and adjusting it so that it refers not to those cells directly, but to those cells indirectly via cells E10, E12 and E14 further down the page.

Cell F4 compares cell F3, the reconstruction, with cell F2, the link to the original cell, to provide reassurance that the reconstruction has successfully reproduced the original calculation. The discrepancy should be zero. If OAK generates reconstructions where the discrepancy is not zero, Operis would like to know about them.

INTERPRETING THE RECONSTRUCTION

The original formula, =E3000+E6000+E9000, involved cells that were thousands of rows apart, which makes it hard to review and understand. The reconstruction restates the calculation so that it involves cells that are just a few rows further down the worksheet, which makes it easy to review and understand.

The formula subject to reconstruction was placed in cell F9. As a result, the link to the cell containing that formula, OAK's reconstruction of it, and the discrepancy between them are all positioned in column F of the reconstruction worksheet.

The cells mentioned in the formula subject to reconstruction were in cells E3000, E6000 and E9000. As a result, the links to those cells in the rows 10, 12 and 14 of the reconstruction worksheet are all positioned in column E.

The fact that the formula is in a different column from its dependents is hard to notice when the cells are many rows apart, as they are in the original spreadsheet, but very much more obvious from the reconstruction. Whether such a misalignment is intentional or unintended would be an appropriate subject for further investigation in a real model review.

3.16 Reconstructing multi-sheet formulas

MAKING LIFE HARDER

The example we used to introduce OAK's formula reconstruction capabilities made reference to cells that were on the same worksheet as itself. The result was a reconstruction that, while somewhat useful, did not achieve anything that could not be achieved using the mechanisms built into Excel for determining a cell's precedents.

The OAK formula reconstructor becomes more powerful when applied to formulas that mention several worksheets.

EXAMPLE

1 Create a new workbook.

2 If the workbook has fewer than four worksheets, add some more, using the Home | Cells | Insert | Insert sheet command.
3 In cell Sheet1!E3000, type the number 10

4 In cell Sheet2!E6000, type the number 20

5 In cell Sheet3!E9000, type the number 30.

6 In cell Sheet4!F9, type the formula =Sheet1!E3000+Sheet2!E6000+Sheet3!E9000. You should see the value 60.

7 With cell Sheet4!F9 still selected, invoke the OAK Review | Formula | Reconstruct command.

8 A dialog will appear, with lots of options. These will be explained later in this help. For now, accept the defaults, and press OK.

OAK will generate a new workbook, like the one below.

### INTERPRETING THE RECONSTRUCTION

This workbook is laid out exactly the same as the example given for the reconstruction a simple, single-worksheet formula, except that the fact that the three precedent are sited on different worksheets is indicated by the blue banded labels in rows 8, 12 and 16.

The original formula, =Sheet1!E3000+Sheet2!E6000+Sheet3!E9000, involved cells that were thousands of rows apart, and on different worksheets, which means that the facilities for examining a formula's precedents do not work well.

The reconstruction restates the calculation so that it involves cells that are just a few rows further down the worksheet, and gathered together on the same worksheet, which means that they can be examined much more easily.

### 3.17 The context of a reconstruction

#### NOT JUST NUMBERS

Most spreadsheets don't just contain numbers. They also contain text, often on the left of the spreadsheet. These label the results and explain what is going on.
When OAK is reconstructing a formula, it endeavours to reproduce relevant labels in its report, to provide an indication of where the values involved come from, and what purpose they serve in the original calculation.

OAK calls this material, informative but not essential to the working a reconstruction, the context of the precedent cells.

SOME CONTEXT APPEARS ANYWAY

A reconstructed formula refers to cells further down the worksheet, which in turn are linked to the precedents of the formula being reconstructed.

In order to offer some understanding of where those precedent cells appear in the original spreadsheet, OAK provides links not only to the cells mentioned in a reconstruction formula, but to all the cells located in the same row.

So for example, reconstructing a formula =H3, that refers to a cell in the middle of the block below...

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>I</th>
<th>J</th>
<th>K</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Costs</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>raw materials</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>3</td>
<td>transportation</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>transportcosts</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>5</td>
<td>Head office expenses</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

...will cause OAK to show the word "transportation" at the left, and all the values 4 along the row, even though only the one in column H is actually mentioned in the reconstructed formula.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>I</th>
<th>J</th>
<th>K</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Item</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Original</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Reconstructed</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Expenses</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>WORKBOOK: Book1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>WORKSHEET: Sheet1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>transportation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Tip: These items aren't just values that are copies of what appeared in the original spreadsheet. They are formulas that link back to the original spreadsheet. They can be followed back to their source by pressing Ctrl+[ (left square bracket), and returned to by pressing F5, Enter.

Tip: There are no link formulas generated where cells are blank. That's why cells A10, C10, and D10 are blank.

MORE CONTEXT AVAILABLE ON REQUEST

OAK can be instructed to display more of the context of the elements of a formula than just a single row. The option is offered at the bottom of the reconstruction dialog box.

If the context option is turned on, and three lines are asked for, then OAK will show up to three rows either side of each row that includes precedent cells used in the reconstruction.
The additional lines of context are distinguished from the ones needed in the reconstruction by being shown in a fainter, grey typeface.

There are circumstances where the number of grey context lines delivered on a reconstruction report is less than the number asked for.

- Obviously, if a row is near the top of a worksheet, there may not be that many rows above it. In such a case, OAK will show as many rows as are available.
- If two rows that play a role in a calculation are close to each other, there will be fewer than the demanded context rows between them.

So, for example, a reconstruction of the formula \( =F1+F4+F5+F15 \), specifying three rows of context, will lead to a report with this pattern of grey lines (entries in column H are annotations added afterwards):

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Item</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Original</td>
<td>26</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Reconstructed</td>
<td>26</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Unexpected</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

3.18 Reconstructing whole calculations

The examples shown so far involve reconstruction of a single formula. But OAK can actually reconstruct an entire calculation.

For example, a calculation of profit as revenue less costs is a single calculation for these purposes...

<table>
<thead>
<tr>
<th>C7</th>
<th>=C3+C5</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>1</td>
<td>Year</td>
</tr>
<tr>
<td>2</td>
<td>Revenue</td>
</tr>
<tr>
<td>3</td>
<td>Costs</td>
</tr>
<tr>
<td>4</td>
<td>Profit</td>
</tr>
</tbody>
</table>
...and can be shown in its entirety in a reconstruction report.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ITEM</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
</tr>
<tr>
<td>1</td>
<td>Original</td>
<td>-5</td>
<td>4.5</td>
<td>16.95</td>
<td>33.045</td>
<td>53.6255</td>
</tr>
<tr>
<td>2</td>
<td>Reconstructed</td>
<td>-5</td>
<td>4.5</td>
<td>16.95</td>
<td>33.045</td>
<td>53.6255</td>
</tr>
<tr>
<td>3</td>
<td>Discrepancy</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

The rule is simple: to be acceptable to OAK’s Formula | Reconstruct command, a selection must be all or part of just one row of a spreadsheet.

3.19 Timelines in reconstructions

When neighboring cells in a formula are consistent, that is, they are copies of each other, it is generally the case that they are arrayed along some dimension. Most commonly that dimension is time, though it does not have to be.

When there is such a dimension involved, it is often set out along the top of the spreadsheet, in the form of a timeline or other enumeration. Excel recognizes the usefulness of this material in defining what the different calculations are about by providing the ability to

- freeze a few rows at the top of the window (and columns down its side) in which the worksheet is being displayed, so that these stay still when the rest of the worksheet is scrolled
- to define print titles which cause similar repetition of selected rows (and columns) when the worksheet is printed.

OAK similarly recognizes that these items are valuable in interpreting a calculation, and provides the means to include them in a reconstruction. The dialog that appears when the Formula | Reconstruct command is invoked provides the means to specify

- that any rows frozen in the current window should be reproduced at the top of the reconstruction
- that any rows specified in the worksheet’s Print titles should be reproduced at the top of the reconstruction
- no extra rows are wanted at the top of the reconstruction (the default).
Though the frozen panes of a window and print titles of a worksheet can specify both rows and columns, only the row part of the definition is used by OAK. The column part is ignored. (That is why the references to columns are parenthesized above.)

Since these rows are not essential to the reconstructed calculation, but help in interpreting it, they are like the context rows that OAK can also include, and are colored the same grey as those rows. Like those context rows, they are not values, but formulas, which link back to the original spreadsheet and can be followed with Ctrl+{ (left square bracket).

Occasionally a timeline does actually participate in a reconstructed calculation. In such a case it will appear below the reconstructed formula, alongside any other links to precedent cells, as well as along the top as context.

The timeline that is reproduced in a reconstruction report is the one that pertains to the worksheet bearing the formula being reconstructed. The hierarchy of precedents includes links to other worksheets. There is no guarantee that those worksheets associate columns with time periods in the same way, so the labeling implied by the report timeline may not be consistent with any cells that originate from them.

Tip: In general it is strongly recommended that worksheets are consistently aligned as to timelines. Writing formulas in terms of meaningful names, which rely on Excel's automatic row and column matching mechanisms, is a good way to enforce this alignment.

### 3.20 Frozen rows and columns

When OAK generates a reconstruction report, it freezes the panes of the window in which it is displayed, so that the top lines stay visible no matter how much scrolling the rest of the worksheet is subject to. The rows that are frozen are

- the three lines that link to the original calculation, reconstruct it, and calculate any discrepancy
- any timeline or similar enumeration inferred from frozen panes or print titles.
OAK also provides the means to extend this freezing to the leftmost columns of reconstruction report window. It is labelled Freeze columns on the dialog that is presented when the Formula | Reconstruct command is used.

3.21 Reconstructing formulas that use names

Opinion is divided between spreadsheet users who value Excel's ability to express formulas in terms of meaningful names, such as =Revenue-Costs, and others who prefer the traditional coordinate notation, such as =D5-D7.

When OAK's Formula | Reconstruct command encounters names in a formula that it is processing, it automatically removes the names and converts the formula into coordinate notation. The idea is that OAK should provide insight into the processing performed by a calculation, by re-expressing it as simply as possible.

This behaviour is particularly useful for checking one kind of error. The formula =Revenue-Costs is arguably easier to read than =D5-D7; but it will nevertheless give the wrong answer if Revenue or Costs have accidentally been defined so that they refer to the wrong cells. Reconstructing the calculation, and checking that the links that appear in the reconstruction report lead to the right place in the spreadsheet under test, is one way to verify that it is not undermined by misdefined names.

For example, in this spreadsheet the total costs is calculated as =CostsFixed+CostsVariable+CostsOverhead, where those three terms are names that purport to have been allocated to the relevant ranges above. But due to a fault, CostsOverhead has become associated with row 5, rather than row 6 as intended.
The fault is not at all apparent from inspecting the spreadsheet. Reconstructing the total makes it obvious: it shows only two rows 10 and 11, instead of the expected three, and the formula shows the double counting of row 11.

In this particular example, the fault could perhaps be seen if Excel is put into Edit mode (click on the formula, or press F2 while the cell is selected); then Excel highlights the cells involved in the calculation, and uses colors to link them to the relevant terms. But it takes particular concentration to notice that two rows are highlighted while one is not, and to see that the same colors are used for two of the formula terms. And in the case when the cells mentioned are on a different worksheet from the formula doing the mentioning, the precedent highlighting is no help at all.

Some auditors will prefer to check the name definitions systematically, one at a time. For those people, OAK offers the Build Names Database command.

### 3.22 Levels of Reconstruction

#### WHAT’S IN A CALCULATION?

The introduction to OAK’s notion of reconstruction refers to it as applying to “a calculation”. A calculation that is at all interesting and substantial will take more than one formula to conclude. At least, it should do. Certainly, the practice of Operis, and of well-trained analysts elsewhere, is to build up results using a sequence of several short formulas, since they are much easier to understand than a small number of long formulas. Trying to derive a result using a single formula that continues for many lines in the formula bar is considered a Bad Thing.

Applying OAK’s Formula | Reconstruct command will therefore in many cases only tell a fragment of the story of a calculation. What is needed is to look not just at the formula in
the chosen cells, but at the precedents of those cells, and their precedents too, possibly all the way back to the inputs.

OAK can be instructed to do exactly this. It will take a formula, for example =F7-F5; look at the formulas in those two cells, and substitute those into the formula, to get perhaps =F20*(1+F22) - (F30+F31). If these four cells in turn mention other cells, their contents will be substituted into the growing formula. Eventually, when the process stops, OAK will deliver a new workbook bearing the reconstruction report.

Since a formula commonly mentions more than one other cell, the result of substituting precedent formulas into a reconstruction can quickly become large due to fan-out. In practice, OAK can...

LEVELS

The main way that the reconstruction can be controlled is by telling OAK how many levels of precedents to substitute.

- A level 0 reconstruction reconstructs a given formula, but does not do any substitution of precedents. The simple examples used to introduce the concept of reconstruction show this basic level of reconstruction.

- A level 1 reconstruction takes a given formula, and then expands it by replacing the references to its precedent cells by their formulas. In the example above, =F20*(1+F22) - (F30+F31) was a level 1 reconstruction.

- A level 2 reconstruction is expanded to include its precedents' precedents.

The number of levels of precedent substitution that OAK is to make is specified in the dialog box that is presented by the Formula | Reconstruct command.

![Reconstructor Options](image)

Low values here will produce a short reconstruction quickly. High values will produce take longer to produce a more elaborate reconstruction.
3.23 Lots of levels

It is hard to know in advance how many levels of precedent substitution to specify when initiating a reconstruction, particularly with a spreadsheet that is unfamiliar.

- If a calculation is built up over several cells representing separate steps, then a reconstruction of no or few levels of precedents will only disclose a portion of how the calculation works.

- A reconstruction of many levels of precedents can produce a reconstruction too convoluted to offer any useful insight into a calculation.

Typically there is a sweet spot where there is some precedent substitution which improves the utility of a reconstruction, but not so much as to make it overcomplicated.

One way forward this is to ask OAK to generate reconstructions at, successively, level 0, level 1, level 2 and so on, stopping when the right level is found.

OAK can generate these multiple reconstructions automatically by ticking the box next to “Show all levels”.

![Screenshot of OAK reconstruction options](image)
With this option switched on:

The different levels are presented as separate worksheets in a workbook newly generated to hold the reconstructions.

The worksheets are named Level 0, Level 1 etc.

It is quick and easy to flip between the worksheets, using the Pg-Up and Pg-Dn keys, to hunt for the one that is most insightful.

With it switched off:

A new workbook will be generated, containing a reconstruction of the specified level only.

There is just one worksheet named Level x, where x is the specified level

The main reason not to show all levels of reconstruction is that it is slightly faster just to generate one level of reconstruction than to generate several of them.

3.24 Mixed levels

EXPANSION NOT ALWAYS POSSIBLE

Though one might ask for a reconstruction that involves a certain number of levels of precedent substitution, OAK may not deliver it.

The most obvious reason is that there are not that many levels of precedent in the model. For example, no precedent substitution is possible in a reconstruction of the formula =F1+F3+F5 if F1, F3 and F5 contain constants.

The second reason is that some functions mention large ranges. The formula =AVERAGE(A1:Z1000) refers to 26,000 cells. Putting every one of those in a reconstruction would be infeasible. OAK applies special treatment to such functions which in many cases results in their precedents not being expanded to the next level of precedent.

A third reason is that a formula may refer to another workbook. If that workbook is open, OAK will follow the link, and continue the precedent substitution. But if the workbook is not open, OAK will not expand a reference to the cell containing the link.

LEVELS CAN BE MIXED

In the formula =F1+F3+F5, we could imagine that

- F1 contains a constant.
- F3 contains the formula =AVERAGE(A1:Z1000).
- F5 refers to G5, which in turn contains =H5+I5+J5
In such a case, OAK will generate a reconstruction equivalent to

\[=F1+\text{AVERAGE}(A1:Z1000)+H5+I5+J5\]

- The F1 terms will be settled when OAK considers level 0 of the reconstruction. That cell contains a constant, which is not amenable to further precedent expansion.

- The \text{AVERAGE}(A1:Z1000) term will be settled when OAK considers level 1 of the reconstruction, and made the substitution of that term for F3; but since it mentions thousands of cells, OAK will attempt no further expansion.

- The H5+I5+J5 subexpression will be settled when OAK considers level 2 of the reconstruction, and made the substitution of that expression for G5, which is what it put at level 1 in place of F5.

So though this would purport to be a Level 2 reconstruction, it actually has a mixture of terms that were decided at levels 0, 1 and 2.

**IT'S A MAXIMUM**

The control on the dialog box presented by OAK when the Formula | Reconstruct is invoked should be thought of as controlling the maximum number of levels of precedent substitution that will be made. What OAK delivers may contain a mixture of levels within the same reconstructed formula, up to that maximum.

If OAK runs out of opportunities to make precedent substitutions before the specified limit, it will deliver fewer levels of reconstruction than asked for. For example, if OAK can make no further precedent substitutions after Level 4, OAK will deliver a Level 4 reconstruction, even though it might have been asked for a level 8 reconstruction.

**3.25 Left-right consistency**

**BEHAVIOUR**

Though one might ask for a reconstruction that involves a certain number of levels of precedent substitution, OAK will not always deliver it.

One subtle reason for this is that OAK seeks to reconstruct a calculation, which it defines as a single cell, or several neighboring cells in a single row that are left-right consistent, that is, have the same formula through being copies of each other.

OAK enforces this requirement before it starts the reconstruction process. If the cells selected for reconstruction do not contain consistent formulas, OAK will decline to reconstruct them.

It is possible that the initial cells will be left-right consistent, so OAK finds them acceptable for reconstruction, but then discovers that at some point the precedent cells are not consistent. In such cases, the rule is simple: OAK will not deliver a reconstruction that is not left-right consistent. OAK will halt the process of precedent substitution in the affected
branch of the calculation, though it will continue in other branches of the calculation that are consistent.

EXAMPLE

The screenshot below shows a spreadsheet in which all rows are left-right consistent between columns D and G. Each row has been annotated in column I with the formula in column G.
If row 15 is reconstructed, at level 0 the result simply reproduces the calculation's action of combining the fixed and variable costs.

At level 1, the calculation is expanded and shows clearly that the fixed and variable costs in nominal terms are the result of inflating the same costs in unescalated terms.
But if, now, we change cell G14 by remove the dollar sign from the formula, it will no longer be an exact copy of its neighbors, though it will still give the same result.
A reconstruction of row 15 gives the same result at level 0, again simply demonstrating the calculation's action of combining the fixed and variable costs.

But at level 1, the result is different. The term for the fixed costs has been expanded, to show that it is derived by inflating the unescalated version of those costs. But the term for variable costs has not been expanded. It is still referring to the nominal version of those costs, because OAK has detected that further to expand that branch of the formula would result in something that would no longer be left-right consistent, and so cease to be a calculation by its definition.

APPLICATIONS

Often, the delivery by OAK of a reconstruction that stretches less far back through the hierarchy of precedents than expected is the first clue that a calculation relies on unintentionally inconsistent formulas at some point in the process. That inconsistency can be identified:

- by having Excel find inconsistent formulas using Home | Find & Select | Goto Special | Row differences
- by looking for the small green indicators that Excel shows in the upper left corner of cells that appear suspect. (These are of at least some use, but they are easy to miss in a big spreadsheet, and didn't show at all in the example above, because they appear when an inconsistent cell is in the middle of a run of other cells, but not when it is at the end of the run as here.)
- by having OAK map the workbook
3.26 Simplifying reconstructions

MESSY

Because the process of reconstructing calculations involves replacing references to precedents with those precedents' formulas, reconstructions can get long and messy. By messy, we mean that the process of substitution can give rise to expressions that no one would write naturally. For example,

- if cell A1 has the formula \( =B2 + C2 \)
- and C2’s formula is \( = - D3 \)
- then the result of substitution will be \( =B2 + - D3 \).

Of course, a more natural formula would be \( =B2 - D3 \).

TRANSFORMATIONS

OAK is capable of addressing this problem by applying a range of mathematical transformations to simplify reconstructed formulas as they are developed. The process is identical to the one used by OAK's Formula | Optimize command. The transformations used are listed at Simplification rules.

Some of the simplification rules are safe; there are no circumstances under which they alter the meaning of a formula. Others work in most circumstances, but there are obscure conditions under which they will alter the effect of a formula. OAK can be instructed to

- optimize the reconstructed formula aggressively, that is, to apply all the simplification rules
- optimize the reconstructed formula conservatively, that is, to apply only the simplification rules that are safe in every circumstance
- not to do any optimizing.
If OAK delivers a reconstruction that shows a non-zero discrepancy, the first thing to try is a move from aggressive to conservative optimization.

**LENGTH**

These transformations don't only control the cosmetics of a reconstruction. They are also helpful in taming the length of the resulting formula, which may become extreme due to fan-out of the precedents whose formulas are substituted.

For example,
- if cell A1 contains the formula 
  
  \[ =\text{IF}(B2<0,0,B2) \]
- and cell B2 contains the formula 
  
  \[ =C3+D3+E3+F3+G3+H3+I3+J3+K3+L3 \]
- then the reconstructed formula, after precedent substitution, will be 
  
  \[ =\text{IF}(C3+D3+E3+F3+G3+H3+I3+J3+K3+L3<0, 0, C3+D3+E3+F3+G3+H3+I3+J3+K3+L3) \]

But one of the transformation rules applied by OAK is that 

\[ =\text{IF}(a>b,a,b) \]

becomes

\[ \text{MAX}(a,b). \]

Applying this turns the formula into 

\[ =\text{MAX}(0, C3+D3+E3+F3+G3+H3+I3+J3+K3+L3), \]

which is very much shorter.

**3.27 Pruning reconstructions**

**CONFINE TO ACTIVE PATH**

Reconstructing a formula 

\[ =\text{CHOICE}(A1,B2,C3,D4,E5,F6) \]

will result in OAK digging back through the precedent hierarchy of all six of the cells mentioned, and producing a potentially long reconstruction formula.
Sometimes that is exactly what is wanted. But more often, it's not useful to have a reconstruction showing the details of all the paths, when only one is active. The point of a reconstruction is to elucidate the operation of a calculation, and that is more effectively done with all superfluous detail pared away.

OAK can do this. It provides facilities for stripping out of reconstruction formulas all references to cells that are not actually active at a particular moment. The action is identical to the expression pruning offered by OAK's OAK Review | Formula | Prune command, and the relevant option in discrepancy analysis. It is the most powerful weapon against the tendency of fan-out to cause reconstructions to swell beyond an insightful size.

OAK reconstructs a calculation, which it defines to be neighboring cells with identical formulas. Two formulas that are identical before pruning may not be identical after pruning: one may take the TRUE branch of an IF(...) while the other takes the FALSE branch. For this reason OAK's facilities for pruning a reconstruction, labelled "Confine to Active Path", are enabled only when the range selected for reconstruction is a single cell.

**SPECIAL CASES OF RECONSTRUCTION PRUNING**

Often, the calculation underlying the build up of a result in a model is simple. Revenues and costs are typically built up as quantity x unescalated price x inflation escalator. Yet the formulas that implement this calculation can be complicated, as they switch between different scenarios, look up the relevant inputs in tables, and cause revenues and costs to start and stop, or ramp up and down, according to specified timing. Switching on the pruning can often deliver reconstructions of the essential, underlying calculation.

Where OAK is asked to reconstruct a formula involving HLOOKUP and VLOOKUP without pruning, the entire lookup table will be shown in reconstruction. That may well be useful if the goal is to check the working of the lookup. With pruning turned on, just the cell looked up will appear in the reconstruction formula. Similar considerations apply to INDEX and LOOKUP.

The OFFSET and INDIRECT functions are unusual in that the cells they resolve to need not have anything to do with the cells mentioned as function arguments.
=OFFSET(A1,1000,0) refers to cell A1001, not to cell A1. OFFSET could refer to any cell on a worksheet; INDIRECT could refer to any cell on any worksheet. (Operis considers them dangerous, and instructs its analysts not to use them.) There is a high chance that a reconstruction of a formula that uses either of these functions won't work, as evidenced by delivery of a report with a non-zero discrepancy. Options in such cases are

- to specify pruning when initiating the reconstruction
- to turn up the number of lines of context that are displayed in the report, in the hope that that will cause the offset rows to appear on the reconstruction report among the lines of context
- to use OAK’s Formula | Prune command on the original model (keep a copy of it!) to prune out the OFFSET or INDIRECT functions before starting the reconstruction.

This last option leads to a more general point: it is often insightful to use OAK’s Formula | Prune and Formula | Reconstruct commands together:

- optionally, prune away some complexity from a spreadsheet using OAK Review | Formula | Prune
- reconstruct the result
- then do further pruning of the reconstruction report.

### 3.28 Reconstructing SUMs

EXPANDING SUMS

When reconstructing, OAK replaces references to cells with the formulas in those cells, back to a number of levels of precedent specified by the user. But it doesn't pursue the process where it encounters functions that aggregate (derive a single value from) large ranges.

An exception is made for SUM, since this is the most common function used in spreadsheets.

OAK offers the option to expand this function. By expand, we mean replace the notation =SUM(A1:A3) by =A1+A2+A3. Once so expanded, the individual elements can have their precedents followed back an arbitrary number of levels.

A limit is placed on this action so that the resulting formula is not infeasibly large. It would be inappropriate to try to expand =SUM(A1:Z1000), because it would list 26,000 cell addresses, far more than can be fitted in a formula.
CONFINING RECONSTRUCTION TO SUMS

A further special treatment for addition is that OAK can be instructed to perform precedent substitution only to the extent that the result is a formula that contains +, - and SUM. This is specified by selecting "Confine to additions" on the dialog that is presented when OAK's Formula | Reconstruct command is invoked.
This action has a number of uses in extracting from financial statements, which have many subtotals provided for presentational reasons only, a report of the underlying detail devoid of those subtotals.

Original spreadsheet

<table>
<thead>
<tr>
<th>Original spreadsheet</th>
<th>Reconstruction of Increase in retained earnings, confined to additions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Revenue</td>
<td>Revenue</td>
</tr>
<tr>
<td>Cost of sales</td>
<td>Cost of sales</td>
</tr>
<tr>
<td>Gross margins</td>
<td>a</td>
</tr>
<tr>
<td>Cash operating costs</td>
<td>d</td>
</tr>
<tr>
<td>EBITDA</td>
<td>e = c - d</td>
</tr>
<tr>
<td>Depreciation</td>
<td>f</td>
</tr>
<tr>
<td>PBIT</td>
<td>g = e - f</td>
</tr>
<tr>
<td>Interest</td>
<td>h</td>
</tr>
<tr>
<td>PBT</td>
<td>i = g - h</td>
</tr>
<tr>
<td>Tax charge</td>
<td>j</td>
</tr>
<tr>
<td>PAT</td>
<td>k = i - j</td>
</tr>
<tr>
<td>Dividends</td>
<td>l</td>
</tr>
<tr>
<td>Increase in retained earnings</td>
<td>m = k - i</td>
</tr>
</tbody>
</table>

Revenue   a  Revenue   a
Cost of sales b  Cost of sales b
Gross margins c = a - b
Cash operating costs d  Cash operating costs d
EBITDA e = c - d
Depreciation f  Depreciation f
PBIT g = e - f
Interest h  Interest h
PBT i = g - h
Tax charge j  Tax charge i
PAT k = i - j
Dividends l  Dividends l
Increase in retained earnings m = k - i

It is also useful for turning cash flow statements that start with EBITDA or some other measure of earnings, into versions that start with Revenue.

If the calculation selected for reconstruction is not itself composed of + or - operators or the SUM function, the result is a rather useless report linking back to the calculation itself.

3.29 Selections suitable for reconstruction

THE REQUIREMENTS

To be acceptable to OAK’s Formula | Reconstruct command, a selection must be
- all or part of just one row of a spreadsheet
- be left-right consistent, that is, contain formulas that are all copies of each other.

In addition, if OAK is asked to reconstruct only the paths in a formula that are active, then what is acceptable becomes more restricted, though it is just a special case of the foregoing requirements: the selection must consist of a single cell.
It is permissible for the cells being reconstructed to form an array, so long as the entire array is selected (and the above requirements have also been met).

MOTIVATION

These requirements are all driven by the same consideration: reconstructing formulas is a potentially expensive thing to do computationally, particularly where formulas have a large degree of fan-out.

To deliver the reconstructions in a reasonable time, OAK does not reconstruct every cell in the selection. Instead, it reconstructs the first cell and the last cell in the selection only, and fills in the intervening cells by copying the reconstructed formulas.

In principle, having established that a selection is left-right consistent, it would be sufficient to reconstruct just the first cell, and to fill in the others by copying the resulting formula. OAK reconstructs the first and last cell, and checks that they give the same formulas (in RC notation). Only if they match does OAK deliver the requested reconstruction. This is consistent with Operis’s strong belief that the result of a calculation is more dependable if it can be derived using two independent methods.

If the two calculations don't match, OAK will report that it has encountered an error. Many months of testing have passed since OAK reported such a mismatch. Nevertheless, since such errors are the result of incorrect programming, Operis would like to know about any that users encounter, so that it can remove them from future versions of OAK.

When OAK is instructed to reconstruct only the active paths in a formula, it prunes the expressions it encounters. There, for example, =IF(condition,truepart,falsepart) will be turned into =truepart, or =falsepart, depending on the result of evaluating condition. In this way, depending on the inputs, two neighboring cells that have identical formula before the pruning may have different formulas after the pruning. OAK would therefore no longer be able to take the short cut of reconstructing only the first and last formulas selected and relying on copying those to fill any intervening cells. For this reason, OAK will only offer the option to limit the reconstruction to the active paths in a formula if the selection consists of a single cell.

3.30 Long formulas

Even with the mechanisms offered by OAK to tame fan-out, the formulas that result from OAK can become very long.

Sometimes they get so long that they pass the limit that Excel is capable of handling. (1024 characters in workbooks for Excel 2003, 8192 characters in Excel 2007)

In such cases OAK places a single quote in front of the result of a reconstruction, so that Excel will treat it as a text constant rather than a formula. But the formula is still real, in the sense that it would work if the quote was removed and Excel could handle the formula. This is easy to demonstrate: Excel 2007’s limit on the length of a formula is higher than
earlier versions of Excel, so it can be used to test reconstructions too long for Excel 2003 workbooks.

A formula that is very long would be of questionable value in a real model. In fact, we have emphasized that the style of Operis, and of well-trained analysts elsewhere, is to build calculations up in simple steps using a number of short formulas. But the purpose of a reconstruction is not to produce shippable code to paying customers. A reconstruction rapidly produces an abstract from a larger model, which stands a chance of shedding insight into what a particular calculation does and how it goes about it. They are disposable: the majority of reconstructions are thrown away a few minutes after they are generated.

3.31 Uses for reconstructions

UNDERSTANDING CALCULATIONS

The most common use for the OAK Review | Formula | Reconstruct command is to understand quickly how a spreadsheet arrives at a calculation, and to extract that information quickly free from the distraction of many other calculations that are surrounding one of interest in large spreadsheet. It performs a function similar to, but more powerful than, Excel's built-in Formulas | Formula auditing | Evaluate formula command.

PARALLEL RECONSTRUCTIONS

Operis believes strongly that the most powerful way to verify a spreadsheet calculation is to demonstrate that the same answers can be derived using a completely different method.

OAK’s Formula | Reconstruct command can certainly help with this. It generates a worksheet that is suitable as a framework for a parallel reconstruction, and it can give a high-level view of what the calculation is doing, and what inputs are being used.

However, it is important to resist the temptation to use the OAK output as the parallel reconstruction. Parallel reconstructions are only powerful if they are orthogonal, that is, go about the calculation in an intentionally different fashion. Reconstructor reports from OAK are merely mechanized restatements of the original calculation, and restate the same logic. It is for the spreadsheet reviewer to use the OAK report as a useful starting point for the reconstruction, and to adapt so that it is indeed different in approach from the original calculation.

ASSESSING SPREADSHEET COMPLEXITY

OAK includes a command for summarising workbooks, since this is a standard offering in products in the same general class as OAK. However, Operis has kept careful records of the time taken to complete hundreds of model audit reviews, and finds that it has very little to do with the number of formulas in a spreadsheet, or how complex it is. The R-squared
between the number of man hours consumed in an assignment and the number of distinct formulas in a spreadsheet was observed to be less than 0.2.

In working out how to staff, schedule and price a model review assignment, Operis finds OAK reconstructions can be more illuminating. A few moments having OAK deliver a reconstruction of the key revenue and cost calculations in a spreadsheet can quickly give an impression of whether the reconstruction reports consist of hundreds or even thousands of lines of material, or merely a handful of them. It turns out that an experienced reviewer can use these to reach a more reliable estimate of the effort needed to prove a spreadsheet than can be gained from the spreadsheet statistics.

### 3.32 Difference between reconstruction and discrepancy analysis

<table>
<thead>
<tr>
<th></th>
<th>RECONSTRUCTION</th>
<th>DISCREPANCY ANALYSIS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Principal purpose</strong></td>
<td>To understand the composition of a single calculation.</td>
<td>To identify differences between two calculations that are supposed to give the same result.</td>
</tr>
<tr>
<td><strong>Input</strong></td>
<td>A single cell, or a number of consecutive cells in a row that all contain the same formula</td>
<td>Two cells which contain results from the calculations to be compared</td>
</tr>
<tr>
<td><strong>Output</strong></td>
<td>Reconstruction reports at one or more levels of precedent, each on a separate worksheet</td>
<td>A fan-out covering (usually) several levels of precedent, laid out on the single worksheet.</td>
</tr>
<tr>
<td><strong>Non-obvious use</strong></td>
<td>Although OAK’s principle weapon against inconsistent results from similar calculations is the discrepancy analyzer, sometimes useful insight is gained by reconstructing a cell containing the unwanted difference between the two calculations</td>
<td>OAK’s reconstruction capability is valuable as a starting point for preparing a parallel reconstruction of a calculation. But sometimes a discrepancy analysis is useful for the same purpose; since it shows on a single sheet contributors to the calculation that are at different levels of the precedent hierarchy. These can then be cut and pasted into the developing parallel reconstruction.</td>
</tr>
</tbody>
</table>
Tip: To get the discrepancy analyzer to display the fan-out of a single cell, select it and another cell that is blank, to meet the requirement that discrepancy analysis compares two cells.

### 3.33 Formula denaming

#### BACKGROUND

Opinion is divided between spreadsheet users who value Excel's ability to express formulas in terms of meaningful names, such as \(=\text{Revenue-Costs}\), and others who prefer the traditional coordinate notation, such as \(=D5-D7\).

If you like using names, you will sooner or later encounter a client or a partner who does not like the things. And if you don't use names in your own work, in time you will be confronted by a spreadsheet that does make use of them.

OAK provides for this situation by providing the means to remove the names in Excel formulas and replace them with traditional coordinate references. Because the action is in effect the reverse of the Formulas | Defined names | Defined name | Apply Names command, which is described in the Lesson E of the tutorial on Names, it is known as the Deapply names command.

#### DETAIL

Removing names from a formula is easy. All one has to do is replace every instance of a name with its definition. Consider a workbook in which:

- a formula in cell Sheet1!K12 reads \(=\text{Revenue} - \text{Costs}\)
- a name Revenue is defined as Sheet1!$D$5:$M$5
- a name Costs is defined as Sheet1!$D$7:$M$7.

In this case, the formula \(=\text{Revenue} - \text{Costs}\) can be rewritten as \(=\text{Sheet1!D5:M5} - \text{Sheet1!D7:M7}\). And that rewritten formula will work perfectly well. (If you don't understand quite why, read the description of Excel's row/column matching process described in Lesson F and Lesson I of the Names tutorial in this help.)

There is no circumstance under which a simple replacement of names by their definitions won't deliver exactly the same calculation as the version of the formula with names. However, the problem with this rewritten formula is that it is far from idiomatic. No one would write a formula like that. They natural thing to write is \(=K5-K7\).

When OAK removes names, it goes to considerable trouble to produce natural formulas. Once it has transformed the original \(=\text{Revenue} - \text{Costs}\) into \(=\text{Sheet1!D5:M5} - \text{Sheet1!D7:M7}\), it further analyses the result to determine whether

- it can dispense with the worksheet names: it can in this case, because the formula is located on the same worksheet as the cells it is acting on.
Excel will engage its row/column matching mechanism: it will in this case, so OAK picks out the cells that would be the result of the match.

The result in this case would be =K5-K7, the formula that comes to mind most naturally. In this way OAK delivers formulas which are natural and readable, and like the ones that an experienced analyst would write.

In addition, OAK Development | Names | Deapply really is symmetric with Excel's Formulas | Defined names | Defined name | Apply Names.command, in that it will round-trip: one can

- take an ordinary formula, such as =D5-D7 in the above example,
- use Formulas | Defined names | Defined name | Apply Names to turn it into a names-based equivalent =Revenue-Costs
- then use OAK Development | Names | Deapply to get =D5-D7 again

One does not get =Sheet1!$D$5:$M$5 - Sheet1!$D$7:$M$7, a mess full of sheet names and dollar signs that relies on column matching to get the answer first thought of.

FURTHER DENAMING

The technologies that underpin OAK's Deapply names command are also used elsewhere in OAK, when reconstructing calculations and performing discrepancy analysis. These functions require OAK to work out which cells are involved in a sequence of calculations, whether they are referred to by simple coordinate reference or by name. OAK needs to anticipate Excel's row and column matching behaviour, so that it can pin down the cells involved exactly (D5 and D7 in the above example), rather than to something near but not quite exact (D5:M7-D5:M5 in the same example.)
Spreadsheet quality
4.1 Risk analysis

QUALITY MATTERS MORE THAN QUANTITY

New with OAK 4.3 is the ability to assess a workbook not just by the quantity of formulas it contains, but by a metric based on its quality. This metric is delivered on the risk analysis, a report delivered as part of a workbook or worksheet summary.

A spreadsheet is considered good if it is free from various practices that Operis considers risk-prone. This section describes those practices, sets out the argument for why they are to be avoided, and explains what can be done to avoid them.

4.2 Nesting

NESTING

In its work as the leading auditor of financial models, Operis has found that the effort needed to get a spreadsheet accurately to reflect a proposed transaction is dictated only a little by how many formulae it contains. What matters much more is how straightforward those formulae are.

Simple, clear formulas are good in themselves. They are also an indicator of the skill of the analyst who developed the model, and of his ability to react effectively and constructively to the audit reports.

The risk analysis assesses the simplicity of a spreadsheet's formulas by counting how many portions of each formula involve nesting of functions or algebraic operators to a depth of 3 or more, 5 or more or 7 or more.

On the metric that measures nesting to a depth of 3, the formula

$$\text{MAX}(0, \text{MIN}(100, \text{LOG}(\text{ABS}(B1)))) + \text{MAX}(0, \text{MIN}(100, \text{LOG}(\text{ABS}(B2))))$$

will have a score of 2, because it contains two parts nested to that depth, the references to B1 and the B2.

RELEVANT TRADE-OFFS

If a calculation is complex, most seasoned spreadsheet modelers would argue for breaking it into short pieces, each on a separate line. Excel supports spreadsheets containing over a million rows, so using a few of them to show the intermediate workings in a long calculation is not consuming a resource that is at all scarce. The visibility of the intermediate workings is helpful in understanding and debugging the calculation.

Here quantity and quality trade off against each other. A model which has many short and simple formulas is to be preferred over one that has fewer, more complex ones.

Not everyone agrees with this view. Spreading a calculation over many spreadsheet cells will tend to make the layout too cluttered to use as the main output, leading the model developer towards a separate report that lays out the results pleasingly. Many like this
clear delineation between the workings and outputs, but there are those who argue that a model is more easily understood if they are not separated.

John Raffensperger, a researcher in New Zealand, feels that spreading a calculation over more cells is verbose and increases the probability of error.

"More cells [may] increase the probability of error. No art prefers verbosity/ [Nesting] should stop when the formula begins to lose readability. The difference of opinion lies in when the loss of readability begins. It is true that longer formulas are less readable, but the cost of a shorter formula can be more cells and a longer precedence tree, which may be less readable."

4.3 OFFSET and INDIRECT

LATE BINDING FUNCTIONS

Excel knows which cells the formula =A1001+A1002+A1003 refer to the moment it is typed in. In =OFFSET(A1001,A1002,A1003), that determination is intentionally deferred to the moment the worksheet is calculated, so that it can be made dependent on the values in relevant spreadsheet cells.

This point is reflected in the fact that Excel's auditing tools give the expected results for the first formula, but nothing useful for the second. It also means that spreadsheets using OFFSET, or its cousin INDIRECT, take longer, sometimes much longer, to calculate, because more processing has to be done at calculation time.

OFFSET could refer to any cell on a worksheet; INDIRECT could refer to any cell on any worksheet. Operis considers the two functions dangerous, and instructs its consultants not to use them. Excel does not make adjustments to the formula that may be necessary when rows or columns are inserted or cells are moved. The resulting brittleness in the face of change is the reason that they are given very high weights in the risk analysis.

There is never a reason to use OFFSET, as there is nothing that can be achieved with it that can’t be achieved more dependably with INDEX() or INDEX():INDEX(). There is sometimes a reason to use INDIRECT, as it allows the deferral to calculation time of identification of which workbook and worksheet an input is to be taken from, something that can be achieved no other way. When the source workbook and worksheet are known, it is normally possible to find other ways to soft code the selection of inputs that don't have INDIRECT's dangers.

RELEVANT TRADE-OFFS

Some organizations are more tolerant of OFFSET and INDIRECT than Operis and would not be so aghast at finding them in a spreadsheet. Users of OAK in such environments can reduce the weight attached by the risk analysis to these functions.
4.4 Obsolete lookups

LOTUS LEFT Overs

The Excel functions VLOOKUP and HLOOKUP are widely used to extract values from tables given a key that indicates which element is wanted. In Operis’s view, these functions are provided for back-compatibility for spreadsheets developed under Lotus 1-2-3, and have no role in newly developed models. They require specification of a row or column offset, which tends to be entered as a fixed value, with the effect that Excel does not make adjustments to the formula that may be necessary when rows or columns are inserted. Operis recommends =INDEX(… , MATCH()) as the way to look things up that combines robustness to model alteration with ease of reading.

Excel’s LOOKUP formula does not suffer from the shortcomings of VLOOKUP and HLOOKUP but =INDEX(… , MATCH()) is to be preferred here also so that all lookups are performed using a single construct.

RELEVANT TRADE-OFFS

The opinion that VLOOKUP and HLOOKUP are historic relics that should be retired is clearly not shared by everyone, as they are widely taught and cited by those who have been so indoctrinated. Users of OAK in such environments can reduce the weight attached by the risk analysis to these functions.

4.5 IFERROR

TRAPPING ERRORS, OR HIDING THEM?

IFERROR allows a spreadsheet developer to anticipate the possibility that a calculation might result in an error value, and to provide an alternative value for such circumstances.

The risk is that IFERROR traps errors of all kinds, not just the one that the developer foresaw. It may mask serious malfunctions of the spreadsheet that would have been better brought to attention through the display of a sea of error values.

4.6 Hardwired constants

HARD TO CHANGE

It’s widely considered poor practice to hard-code assumptions into formulas, such as =A1/7. Here, the 7 is the life in years over which an asset is to be depreciated. Exploring the effect of switching to an 8 year asset life requires altering formulas, and it’s hard to be sure that every relevant formula has been altered. It’s better to put the assumption in a cell, and refer to that in the formulas. Then the assumption can be flexed reliably by changing a single value.
RELEVANT TRADE-OFFS

Some constants are so well recognised that it is hard to insist that they can’t appear in a formula: 12 (months in year), 365 (days in year).

4.7 Cell reference notation

MEANING

Excel supports the possibility of expressing formulas both in the coordinate notation customary in spreadsheets, or in terms of quantities with names that are meaningful in the context of the problem, such as Sales or Costs. There is a range of opinions and current practices on how much use should be made of this capability.

An impression can be formed of where the author of a spreadsheet model sits in this debate by the counts set out in the risk analysis of the terms each formula, allocated between

- Unmatched names: references by name to one or more cells which Excel can involve in the calculation of a formula simply by substituting the equivalent coordinates
- Matched names: references by name to number of cells which leaves Excel to pick out the single cell to be used in a calculation, on the basis of matching the row and/or column in which the formula lies
- Cell references: references in coordinate style to one or more cells

The distinction between matched and unmatched names is explained in more detail in this manual’s tutorial on Excel names.

RELEVANT TRADE-OFFS

Operis values the opportunity to express formula using meaningful names. For this reason it gives a negative weight to the appearance of name references in a model, signalling its opinion that their use reduces the overall risks arising from the spreadsheet. Not everyone thinks it is wise to use Excel’s ability to reference dependents by name in a formula, or understands how to. OAK users in this camp are free to alter the weighting to a positive value of their choice to identify spreadsheets that are expressed in a way they find undesirable.

4.8 Cell reference profligacy

SLUGGISH EXCEL

Repeated references to huge ranges of cells where much smaller ones will work just as well can cause Excel’s performance to become sluggish, which makes them hard to
Reference material: Spreadsheet quality

check. Often the references are the results of mistakes. To gauge this, the risk analysis lists the number of cells referenced in a formula.

There are occasions when this quantity is not easily determined. References that mix absolute and relative elements, for example \( =\text{SUM}($A1:C1) \), evaluate to ranges of different size as they are copied. Because it changes from one instance of the formula to the next, OAK can’t distil a count of the number of cells used into a single number.

For the references that it can size, because they are consistent in all copies of the formula, OAK places a count of the cells mentioned in the column headed Cells Referenced. For the references it can’t easily size, because they refer to items of variable size, OAK places a count of the references in the formula in the neighboring column, labelled Variable ranges.

For example, in the formula \( = \text{SUM}(A1:A5) + \text{SUM}($B$1:$B$5) + \text{SUM}($C1:C5) \), Excel will

- place 10 in the Cells Reference column, being the number of cells mentioned in the references \( A1:A5 \) and \( $B$1:$B$5 \)
- place 1 in the Variable References column, being the number of references that mix absolute and relative addressing, here \( $C1:C5 \).

RELEVANT TRADE-OFFS

The presence of a mixed-reference term in a formula can be the result of a simple mistake.

Even where it is not a mistake, a mixed reference term merits careful examination. A cell that is a copy of its neighbour has a reasonable chance of being correct if its neighbour is known to be correct. But this is not so true for cells using mixed references, as their meaning changes as they are copied. There are normally other ways of coding for the same effect without using mixed absolute and relative addresses, though they may consume three or four spreadsheet rows. The advantage of the mixed addressing is that a calculation can often be accomplished in a single row.

4.9 Error constants

ERROR CONSTANTS

Besides numeric constants such as 1, 2 and 345.6, and text constants such as “Hello, World”, Excel permits a handful of other constants to be present in a formula. Two are the boolean constants TRUE and FALSE. The rest are the error values, #N/A!, #DIV/0, #REF!, and #NUM! The risk analysis provides a count of the occurrences of these error terms.
RELEVANT TRADE OFFS

Any of the error values can be included in a formula and there exist rare circumstances where it is meaningful and useful to do so. Most of the time, their presence in a formula is the result of an error. The most common example is when a formula mentions a range that is later deleted; Excel changes the reference to #REF!. 
OAK object reference
**5.1 OAK object reference**

This section lays out the OAK object model, which is exposed for scripting. All the interfaces, methods, properties and enumerations are listed in alphabetical order.

The key interfaces in the model are IOAKAddIn and IOAKAPI.

**5.2 AnalyzeDiscrepancies**

Generates a new worksheet which sets out an analysis of the precedents of two cells, inspection of which may be helpful in identifying the cause of any differences between the two cells' values.

**APPLIES TO**

IOAKAddIn, IOAKAPI

**SYNTAX**

result = expression1.AnalyzeDiscrepancies

Set wb = expression2.AnalyzeDiscrepancies(range1, range2, levels, linkByValue, decimals, suppressNonMatched, maxCells, PruningMode)

- **result**  An OAKResult enumeration indicating the success or otherwise of the action.
- **wb**  A workbook generated by OAK on which the discrepancy analysis is laid out.

- **Range1**  Required Range. The first of the two cells whose derivations are to be compared. Can be a Range object or a string that contains a cell reference in R1C1-style notation.

- **Range2**  Required Range. The second of the two cells whose derivations are to be compared. Can be a Range object or a string that contains a cell reference in R1C1-style notation.

- **levels**  Required Integer. The number of levels of precedent that the analysis is to include.

- **linkByValue**  Required Boolean. False to cause precedent cells that are common to both starting cells to be highlighted. True to cause precedents that are common to both starting cells to be highlighted if they have the same value, but are not necessarily located in the same cell.
**decimals**  Required **Integer**. The number of decimal places to compare when **linkByValue** is TRUE. Ignored when **linkByValue** is FALSE.

**suppressNonMatched**  Required **Boolean**. **False** to display the trees of both starting cells’ precedents in their entirety. True to suppress the display of subtrees that have no common elements between the two fan-outs.

**maxCells**  Required **Integer**. Controls the potentially explosive fan-out of a precedent hierarchy by limiting the number of cells that will be included in the analysis in any large range referenced by a function. If a cell encountered during the discrepancy analysis contains the subexpression =SUM(A1:A1000), and maxCells is set to 20, only the first 20 cells in the range A1:A1000 will be explored in the analysis, not all 1000 of them.

**PruningMode**  Optional **PruningMode**. Specifies what, if any, pruning is performed on functions encountered during the analysis of precedent formulas.

**REMARKS**

When applied to an **IOAKAddIn** object, the AnalyzeDiscrepancies method activates the same dialog box as is presented when the Analyze discrepancies command is selected manually from the OAK user interface. The action that follows is controlled by the data gathered by the dialog box from the user.

When applied to an **IOAKAPI** object, the AnalyzeDiscrepancies method generates a discrepancy analysis identical to the one that would be generated by OAK if its Analyze Discrepancies command were activated manually from the user interface, but the details of the analysis are specified by the parameters submitted to the method.

**EXAMPLE**

To analyse the differences in derivation between two cells MyRange1 and MyRange2, back to 4 levels of precedence, pruning occurrences of the COUNTIF and SUMIF functions

```vba
Set MyWorkbook = MyOakApi.AnalyzeDiscrepancies MyRange1, MyRange2, 4, False, 0, True, 500, PruningMode_COUNTIF + PruningMode_SUMIF
```

5.3 **ApplyNames**

Applies the specified names to the specified range.

**APPLIES TO**

**IOAKAddIn** , **IOAKAPI**

**SYNTAX**

```
result = expression1 .ApplyNames
```
expression2 .ApplyNames(Names, Range)

expression1 Required. An expression that returns an IOAKAddIn object.

expression2 Required. An expression that returns an IOAKAPI object.

result An OAKResult enumeration indicating the success or otherwise of the action.

Names Required Variant. The names to apply. Can be an array of strings or a collection containing strings.

Range Required Range. The range to apply the names to.

Scope Optional Variant. Can be one of Range, a Worksheet, a Workbook, or missing, in which case the whole workspace is presumed.

REMARKS

When applied to an IOAKAddIn object, the ApplyNames method activates the same dialog box as is presented when the Apply names command is selected manually from the OAK user interface. The action that follows is controlled by the data gathered by the dialog box from the user.

When applied to an IOAKAPI object, the ApplyNames method applies the specified names over the range indicated by the parameters submitted to the method.

5.4 BuildNameDatabase

Generates a new workbook or worksheet which sets out a table of details of the names defined in a workbook.

APPLIES TO

IOAKAddIn, IOAKAPI

SYNTAX

result = expression1 .BuildNameDatabase

Set wbOut = expression2 .BuildNameDatabase(wbIn, hiddenNames, specialNames, excludeSpecialNames, outputInSameWorkbook, findOverlaps)

expression1 Required. An expression that returns an IOAKAddIn object.

expression2 Required. An expression that returns an IOAKAPI object.

result An OAKResult enumeration indicating the success or otherwise of the action.
**wbOut**  A workbook generated by OAK on which the discrepancy analysis is laid out.

**wbIn**  The workbook containing the names that are to be reported by OAK.

**hiddenNames**  Required **Boolean**. TRUE if the database is to include names that are hidden or very hidden. FALSE if the database is to include visible names only.

**specialNames**  Required **Variant**. Can be Nothing, or an **Array** or **Collection** of **Strings**, specifying any names that are to be excluded from the database if **excludeSpecialNames** is True. These would normally be names that have particular meaning to Excel.

**excludeSpecialNames**  Required **Boolean**. TRUE if the database is to exclude from the report of overlapping names any names that are members of the list of special names provided to it. FALSE if it is to show all names.

**outputInSameWorkbook**  Required **Boolean**. TRUE if the database is to be laid out on a worksheet to be newly added to **wbOut**. FALSE if the database is to be laid out on a worksheet in a new workbook.

**findOverlaps**  Required **Boolean**. TRUE if the database is to be supplemented by a report of names that overlap, and so are potential duplicates or sources of error. FALSE if the report of overlapping names is unwanted, which is significantly faster, particularly if the number of defined names is large, since every names has to be compared with every other.

**REMARKS**

When applied to an **IOAKAddIn** object, the BuildNameDatabase method activates the same dialog box as is presented when the Analyze discrepancies command is selected manually from the OAK user interface. The action that follows is controlled by the data gathered by the dialog box from the user, and acts on the currently active workbook.

When applied to an **IOAKAPI** object, the BuildNameDatabase method generates a report of defined names identical to the one that would be generated by OAK if its Build Name Database command were activated manually from the user interface, but the details of the analysis, including the workbook to be examined, are specified by the parameters submitted to the method.

**EXAMPLE**

To report the names defined in the ActiveWorkbook, as quickly as possible (and so omitting the overlapping names test)

```vba
Set MyWorkbook = MyOakApi.BuildNameDatabase(ActiveWorkbook, FALSE, Nothing, FALSE, FALSE, FALSE)
```
5.5 Compare

Identifies any differences between

- all the worksheets in one workbook all of the worksheets in another
- a subset of the worksheets in one workbook with the equivalently named worksheets in another
- one worksheet with another worksheet

APPLIES TO

IOAKAddln, IOAKAPI

SYNTAX

Set result1 = expression1 .Compare

Set result2 = expression2 .Compare(bookOrSheets1, bookOrSheets2, testFormulas, ignoreCase, styleA1, alignRows, alignColumns, highlightDifferences, writeReport, groupEquivalentChanges, compareNames, rowAlignmentColumns, columnAlignmentRows)

expression1 Required. An expression that returns an IOAKAddln object.

expression2 Required. An expression that returns an IOAKAPI object.

result1: An OAKResult enumeration indicating the success or otherwise of the action.

result2 An ICompareResult object that gives access to the results of the comparison.

BookOrSheets1, BookOrSheet2 Required. The workbook or worksheet(s) to be compared. One of

- an Excel workbook object
- an array of workbook objects
- a collection of worksheet objects
- an Excel worksheet object
- an array of worksheet objects
- a collection of worksheet objects.

testFormulas Required Boolean. True if the comparison is to be made in terms of cell formulas. False if the comparison is to be made in terms of cell values.
**ignoreCase**. Required *Boolean*. *True* if the comparison is to be sensitive to case, so that Dog and dog are reported as different. *False* if the comparison is to be case insensitive, so that Dog and dog are not reported as different.

**styleA1**. Required *Boolean*. *True* if the reports produced by the comparison are to display formulas in A1 notation. *False* if they are to display formulas in R1C1 notation. (The comparison is performed using R1C1 notation; this switch just controls how the differences are presented.)

**alignRows** Required *Boolean* or *CompareAlignment*. *True* if OAK is to attempt to minimize the differences reported between the workbooks or worksheets by inserting blank rows to align the headings in the columns specified in *rowAlignmentColumns* before comparing the cell contents. *False* if OAK is to make its comparison without inserting any rows. More advanced alignment options can be specified using a combination of *CompareAlignment* values.

**alignColumns** Required *Boolean* or *CompareAlignment*. *True* if OAK is to attempt to minimize the differences reported between the workbooks or worksheets by inserting blank columns to align the headings in the rows specified in *columnAlignmentRows* before comparing the cell contents. *False* if OAK is to make its comparison without inserting any columns. More advanced alignment options can be specified using a combination of *CompareAlignment* values.

**writeReport** Required *Boolean*. *True* if OAK is to generate a new workbook summarizing the differences it has found. *False* if OAK is to generate no report workbook.

**groupEquivalentChanges** Required *Boolean*. *True* if OAK is to attempt to minimize the differences reported between the workbooks or worksheets by grouping blocks of equivalent differences, which is likely better for a human reader. *False* if OAK is to report each cell difference separately, which may be better for further machine processing of the OAK output.

**compareNames** Required *Boolean*. *True* if OAK is to create a report of differences between names in the workbooks or worksheets.

**rowAlignmentColumns** Required *Variant*. A string indicating which columns hold the headings to be used for row alignment, if *alignRows* is *True*, in the format "A:B".

**columnAlignmentRows** Required *Variant*. A string indicating which rows hold the headings to be used for column alignment, if *alignColumns* is *True*, in the format "2:3".

**REMARKS**

When applied to an *IOAKAddIn* object, the Compare method activates the same dialog box as is presented when the Compare command is selected manually from the OAK user...
interface. The action that follows is controlled by the data gathered by the dialog box from the user.

When applied to an IOAKAPI object, the Compare method performs a comparison between the workbook or worksheets specified by the parameters submitted to the method.

One might expect the method has nothing to do if both alignRows and writeReport are set to False. But that is not so; it needs to perform a comparison, without alignment, so that result2 (an ICompareResult object) will be defined, and in particular so that result2.TotalDifferences is accurate.

EXAMPLE

The code fragment used as an example in the introduction to scripting shows OAK identifying which spreadsheet in a specified directory is most like the active workbook. It illustrates the use of the Compare and UndoCompareModifications methods, and the ICompareResult interface.

5.6 CompareAlignment

A bitmapped enumeration used to specify the alignment options for IOAKAPI.Compare.

APPLIES TO

Operis_OAK

SYNTAX

Where the Compare function needs to be provided with advanced alignment parameters, it can be given values built up from the following:

- CompareAlignment_None No alignment specified
- CompareAlignment_Inherit Use the same value/formula and case sensitivity as is used for the comparison
- CompareAlignment_AlignFor Align using formulas
- CompareAlignment_AlignVal Align using values
- CompareAlignment_CaseIns Case insensitive alignment
• CompareAlignment_CaseSensitive Case sensitive alignment.

EXAMPLE

To specify that column or rows should align formulas and be case sensitive, use

```vba
CompareAlignment_AlignFormulas + CompareAlignment_CaseSensitive
```

### 5.7 CompareContent

An enumeration used to specify the content to be compared by options for IOAKAPI.CompareRanges and IOAKAPI.CompareRangesWithColors.

APPLIES TO

**Operis OAK**

SYNTAX

Where the various Comparison functions need to be provided with a parameter that specifies the content to be compared, it can be given:

- CompareContent_Values
- CompareContent_FormulasA1
- CompareContent_FormulasR1C1

### 5.8 CompareRanges

Identifies any differences between a pair of ranges.

APPLIES TO

**IOAKAddIn**, **IOAKAPI**

SYNTAX

```vba
Set result1 = expression1 .CompareRanges
```
C        Reference material: OAK object reference

Set result2 = expression2 .CompareRanges(range1, range2, compareContent, ignoreCase, styleA1, highlightDifferences, writeReport, groupEquivalentChanges)

expression1  Required.  An expression that returns an IOAKAddIn object.
expression2  Required.  An expression that returns an IOAKAPI object.
result1:  An OAKResult enumeration indicating the success or otherwise of the action.
result2  An ICompareResult object that gives access to the results of the comparison.
range1 , range2  Required.  The ranges to be compared
compareContent  Required CompareContent.  Specifies if the comparison is to be made in terms of cell formulas (A1 or R1C1), or cell values.
ignoreCase  Required Boolean.  True if the comparison is to be sensitive to case, so that Dog and dog are reported as different. False if the comparison is to be case insensitive, so that Dog and dog are not reported as different.
styleA1  Required Boolean.  True if the reports produced by the comparison are to display formulas in A1 notation. False if they are to display formulas in R1C1 notation. (The comparison is performed using R1C1 notation; this switch just controls how the differences are presented.)
writeReport  Required Boolean.  True if OAK is to generate a new workbook summarizing the differences it has found. False if OAK is to generate no report workbook.
groupEquivalentChanges  Required Boolean.  True if OAK is to attempt to minimize the differences reported between the workbooks or worksheets by grouping blocks of equivalent differences, which is likely better for a human reader. False if OAK is to report each cell difference separately, which may be better for further machine processing of the OAK output.

REMARKS

When applied to an IOAKAddIn object, the Compare method activates the same dialog box as is presented when the relevant Compare command is selected manually from the OAK user interface. The action that follows is controlled by the data gathered by the dialog box from the user.

When applied to an IOAKAPI object, the Compare method performs a comparison between the workbook or worksheets specified by the parameters submitted to the method.
5.9 CompareRangesWithColors

Identical to CompareRanges, but with additional parameters to control the colors that are used in reports.

APPLIES TO

IOAKAddIn, IOAKAPI

SYNTAX

Set result = expression .CompareRangesWithColors(range1, range2, compareContent, ignoreCase, styleA1, highlightDifferences, writeReport, groupEquivalentChanges, win32ModifiedCellColor, win32DeletedCellColor, win32AddedCellColor, win32BlankAlignmentRowColor, win32BlankAlignmentColumnColor, win32ArrayMembershipColor, win32RangeSelectionColor)

expression Required. An expression that returns an IOAKAPI object.

result An ICompareResult object that gives access to the results of the comparison.

range1, range2, compareContent, ignoreCase, styleA1, highlightDifferences, writeReport, groupEquivalentChanges: These parameters are identical to those of the CompareRanges method.

win32ModifiedCellColor, win32DeletedCellColor, win32AddedCellColor, win32BlankAlignmentRowColor, win32BlankAlignmentColumnColor, win32ArrayMembershipColor, win32RangeSelectionColor: Required Longs which specify the colors to be used for when marking comparison results on a worksheet. See the section titled Colors in compare commands for a VBA function that shows how to generate these values.

These colors are equivalent to the ones which may be specified through the user interface by pressing the Set Colors button in the dialog presented when the OAK Review | Compare | Ranges command is invoked.
5.10 CompareWithColors

Identical to Compare, but with additional parameters to control the colors that are used in reports.

- all the worksheets in one workbook all of the worksheets in another
- a subset of the worksheets in one workbook with the equivalently named worksheets in another
- one worksheet with another worksheet

APPLIES TO

IOAKAddIn, IOAKAPI

SYNTAX

Set result = expression .CompareWithColors(bookOrSheets1, bookOrSheets2, testFormulas, ignoreCase, styleA1, alignRows, alignColumns, highlightDifferences, writeReport, groupEquivalentChanges, compareNames, rowAlignmentColumns, columnAlignmentRows, win32ModifiedCellColor, win32DeletedCellColor, win32AddedCellColor, win32BlankAlignmentRowColor, win32BlankAlignmentColumnColor, win32ArrayMembershipColor)

expression Required. An expression that returns an IOAKAPI object.

result An ICompareResult object that gives access to the results of the comparison.

bookOrSheets1, bookOrSheets2, testFormulas, ignoreCase, styleA1, alignRows, alignColumns, highlightDifferences, writeReport, groupEquivalentChanges, compareNames, rowAlignmentColumns, columnAlignmentRows: These parameters are identical to those of the Compare method.

win32ModifiedCellColor, win32DeletedCellColor, win32AddedCellColor, win32BlankAlignmentRowColor, win32BlankAlignmentColumnColor, win32ArrayMembershipColor: Required Longs which specify the colors to be used for when marking comparison results on a worksheet. See the section titled Colors in compare commands for a VBA function that shows how to generate these values.

These colors are equivalent to the ones which may be specified through the user interface by pressing the Set Colors button in the dialog presented when the OAK Review | Compare | Workbooks/Worksheets command is invoked.
5.11 CompareWorkbooks

Activates the Compare Workbooks or Compare Worksheets user interface, in order to do the following:

- all the worksheets in one workbook all of the worksheets in another
- a subset of the worksheets in one workbook with the equivalently named worksheets in another
- one worksheet with another worksheet

APPLIES TO

IOAKAddIn

SYNTAX

Set result = expression .CompareWorkbooks

Set result = expression .CompareWorksheets

expression Required. An expression that returns an IOAKAddIn object.

result: An OAKResult enumeration indicating the success or otherwise of the action.

REMARKS

When applied to an IOAKAddIn object, the CompareWorkbooks and CompareWorksheets methods activate the same dialog box as is presented when the corresponding Compare commands are selected manually from the OAK user interface. The action that follows is controlled by the data gathered by the dialog box from the user.
5.13 **CopyAddress**

Activates the Copy Address user interface.

APPLIES TO

**IOAKAddIn**

SYNTAX

```plaintext
result = expression .CopyAddress
```

- `expression` Required. An expression that returns an **IOAKAddIn** object.
- `result` An OAKResult enumeration indicating the success or otherwise of the action.

REMARKS

The CopyAddress method acts on the current selection in the same way as the Copy Address command does. An exception may be raised if the selection is not suitable for the action.

The equivalent function in the IOAKAPI interface is IOAKAPI.FormatAddress. This function formats the address(es), but returns the resulting string rather than copying it to the clipboard.

5.14 **CopyLiteral**

Copies the formula(s) from the source range to the target range, without adjusting cell coordinates as Excel normally does.

(More accurately: when Excel copies formulas, it preserves them in R1C1 notation, which has the effect of causing them to change when shown in A1 notation. OAK’s Copy literal command preserves them in A1 notation, which has the effect of causing them to change when shown in R1C1 notation.)

APPLIES TO

**IOAKAddIn**, **IOAKAPI**

SYNTAX

```plaintext
result = expression1 .CopyLiteral
```

- `expression2` .CopyLiteral (source, target)
expression1  Required. An expression that returns an \texttt{IOAKAddIn} object.

expression2  Required. An expression that returns an \texttt{IOAKAPI} object.

result  An OAKResult enumeration indicating the success or otherwise of the action.

source  Required \texttt{Range}. The cells to be copied.

target  Required \texttt{Range}. The place where the cells are to be copied to.

REMARKS

When applied to an \texttt{IOAKAddIn} object, the CopyLiteral method acts on the current selection in the same way as the Copy literal command does. An exception may be raised if the selection is not suitable for the action.

When applied to an \texttt{IOAKAPI} object, the CopyLiteral method acts on the ranges indicated by the parameters submitted to the method.

\textbf{5.15 DeapplyNames}

Deapplies the specified names from the specified scope.

\textbf{APPLIES TO}

\texttt{IOAKAddIn , IOAKAPI}

\textbf{SYNTAX}

\begin{verbatim}
result1 = expression1 .DeapplyNames

Set result2 = expression2 .DeapplyNames(names, scope)
\end{verbatim}

expression1  Required. An expression that returns an \texttt{IOAKAddIn} object.

expression2  Required. An expression that returns an \texttt{IOAKAPI} object.

result1  An OAKResult enumeration indicating the success or otherwise of the action.

result2  An object implementing the IDeapplyNamesResult interface whereby any names were not successfully applied may be identified.

names  Required \texttt{Variant}. The names that are to be deapplied, specified as an array or collection of Excel Name objects.

scope  Optional \texttt{Variant}. A range, worksheet or workbook in which to deapply the names. The default scope is the whole workspace.
REMARKS

When applied to an IOAKAddIn object, the DeapplyNames method activates the same dialog box as is presented when the Deapply names command is selected manually from the OAK user interface. The action that follows is controlled by the data gathered by the dialog box from the user.

When applied to an IOAKAPI object, the DeapplyNames method takes opportunities to adjust formulas within the specified scope so that references to cells by names are replaced to ones by cell reference. The method returns an IDeapplyNamesResult which may be examined to determine which names were or were not replaced.

The DeapplyNames method accessed through IOAKAPI does not offer the option to ignore constants that is available in through the user interface. It is left to the programmer to determine the correct list of names to deapply.

EXAMPLE

To report the names defined in the ActiveWorkbook, as quickly as possible (and so omitting the overlapping names test)

```vba
Set MyWorkbook = MyOakApi.BuildNameDatabase(ActiveWorkbook, FALSE, Nothing, FALSE, FALSE, FALSE)
DeleteNamesInCells
```

5.16 DeleteNamesInCells

Deletes any names in the current workbook that coincide with any of the values in the specified range.

APPLIES TO

IOAKAddIn, IOAKAPI

SYNTAX

```vba
expression1 .DeleteNamesInCells
expression2 .DeleteNamesInCells(Range)
```

- `expression1` Required. An expression that returns an IOAKAddIn object.
- `expression2` Required. An expression that returns an IOAKAPI object.
- `Range` Required Range. The range containing the names to be deleted.
REMARKS

When applied to an *IOAKAddIn* object, the DeleteNamesInCells method acts on the current selection in the same way as the OAK Development | Names | Delete command does. An exception may be raised if the selection is not suitable for the action.

When applied to an *IOAKAPI* object, the DeleteNamesInCells method applies the specified names over the range indicated by the parameter submitted to the method.

The DeleteNamesInCells method is included for completeness. It would be unusual to use it from a VBA program. Its usual use is from the user interface, immediately after an unintended or incorrect use of Excel's Apply names command, to remove the potentially large number of unwanted names thereby produced.

EXAMPLE

To delete those names in the active workbook that coincide with any values in column Z:

```vba
Dim o As Operis_OAK.IOAKAddIn
Dim r as range

Set o = CreateObject("Operis.OAK.Connect")
Set o.ExcelApplication = Application

set r=Range("Z:Z")
o.DeleteNamesInCells r
```

5.17 **DeleteRowsOrColumns**

Deletes the rows or columns indicated, even (unlike Excel's Delete command) if they intersect array constants, array formulas or data tables.

APPLIES TO

*IOAKAddIn*, *IOAKAPI*

SYNTAX

```
result = expression1 .DeleteRowsOrColumns
expression2 .DeleteRowsOrColumns(Range)
```

*expression1* Required. An expression that returns an *IOAKAddIn* object.

*expression2* Required. An expression that returns an *IOAKAPI* object.

*result* An OAKResult enumeration indicating the success or otherwise of the action.

*Range* Required *Range*. The range to be deleted.
REMARKS

When applied to an `IOAKAddIn` object, the `DeleteRowsOrColumns` method performs the same action as the Home | Cells | Delete command. It deletes the selected rows or columns, even if they intersect array constants, array formulas or data tables.

When applied to an `IOAKAPI` object, the `DeleteRowsOrColumns` method deletes the rows or columns indicated by the parameter submitted to the method.

WRINKLES

To specify what is to be inserted unambiguously, `Range` needs to be one or more entire rows or one or more entire columns. It must consist of a single area only; a multi-area range will generate an error.

Though the Delete method can delete rows or columns that pass through array constants, array formulas or data tables, it cannot perform deletions that intersect pivot tables.

The Delete method will not work on worksheets that are protected.

5.18 FormatAddress

Formats the address of a range according to specified options. Effectively does the formatting part of the Copy Address function provided by the OAK user interface, but does not perform the side-effect of copying the result to the clipboard.

APPLIES TO

`IOAKAPI`

SYNTAX

```
expression .FormatAddress (source, prefixAllRegions, bookPrefix, sheetPrefix, a1, absolute, delimiterType, extraSpace)
```

*expression*  
Required. An expression that returns an `IOAKAPI` object.

*result*  
An OAKResult enumeration indicating the success or otherwise of the action.

*source*  
Required `Range`. The cells that supply the address. This can be a multi-area range.

*prefixAllRegions*  
Required `Boolean`. Specifies whether to prefix all areas in the source range.
bookPrefix
prefix. Required Boolean. Specifies whether to include the workbook in the

sheetPrefix
the prefix. Required Boolean. Specifies whether to include the worksheet in

a1 (false). Required Boolean. Specifies A1 notation (true) or RC notation

absolute Required Boolean. Specifies whether addresses will be absolute.

delimiter the result. Required FormatAddressDelimiter. Specifies the delimiter used in

eextraSpace delimiter.. Required Boolean. Specifies if a space should be added after the

REMARKS

The FormatAddress function acts on the current selection in the same way as the Copy
Address command does.

The equivalent method in the IOAKAddIn interface is IOAKAddIn.CopyAddress. This
method activates the CopyAddress user interface.

5.19 FormatAddressDelimiter

An enumeration used to specify the delimiter to be used by IOAKAPI.FormatAddress.

APPLIES TO

Operis_OAK

SYNTAX

Where the FormatAddress function needs to be provided with a parameter that specifies
the delimiter to be used, it can be given:

- FormatAddressDelimiter_Comma
- FormatAddressDelimiter_Tab
- FormatAddressDelimiter_CrLf
• FormatAddressDelimiter_SemiColon
• FormatAddressDelimiter_Default (equal to FormatAddressDelimiter_Comma)

EXAMPLE

To format the address of MyRange1 with no prefixes, and separated by tabs, use:

```
address = MyOakApi.FormatAddress(MyRange1, False, False, False, True, False,
                                  FormatAddressDelimiter_Tab, false)
```

5.20 GetVersion

Returns a string indicating which version of OAK is in use.

APPLIES TO

IoAKAddIn, IoAKAPI

SYNTAX

```
result = expression.GetVersion
```

expression  Required. An expression that returns an IoAKAddIn object or an IoAKAPI object.

result  A string containing the version number of the currently running instance of OAK.

EXAMPLE

To display the current version of OAK in the immediate window:

```
Dim o As Operis_OAK.IOAKAddIn
Dim s As String

Set o = CreateObject("Operis.OAK.Connect")
Set o.ExcelApplication = Application

s = o.GetVersion
Debug.Print "OAK version "; s
```

5.21 ICompareResult

Interface that gives access to the results of an OAK comparison of workbooks or worksheets.
APPLIES TO
The *Operis_OAK* object library

PROPERTIES

**ReportBook Workbook.** The workbook that OAK has generated to contain a report of a comparison. May be Nothing if OAK was instructed not to generate a report.

**TotalDifferences double.** The number of differences that OAK has logged in a comparison.

### 5.22 IDeapplyNamesResult

Calls to IOAKAPI.DeapplyNames return an object that can be examined to determine what if any problems the call encountered.

APPLIES TO
The *Operis_OAK* object library

PROPERTIES AND SYNTAX

```
b = expression .Success
```

```
i = expression. FailedRangeCount
```

```
o = expression. FailedRanges
```

*expression* Required. An expression that returns an *IDeapplyNamesResult*; typically a variable which holds the object returned by a recent call to **IOAKAPI.DeapplyNames**.

*b* *Boolean.* TRUE if the DeapplyNames method succeeded in deapplying all the specified names throughout the specified scope. FALSE if the DeapplyNames method did not manage to deapply any of the specified names completely within the scope.

*i* *Integer.* The number of items in the *FailedRanges* array.

*o* *Object.* An *Array* of *Ranges* indicating where the deapply names operation could not be completed.

REMARKS

Common reasons for the DeapplyNames method to fail are

- that a cell in the scope was protected
that it would result in a formula longer than Excel can accept

5.23 InsertRowsOrColumns

Inserts rows or columns at the point indicated, even (unlike Excel's Insert command) if they intersect array constants, array formulas or data tables.

APPLIES TO

IOAKAddln, IOAKAPI

SYNTAX

\[
\text{result} = \text{expression1} .\text{Insert} \text{expression2} .\text{Insert}(\text{Range, copyFormula})
\]

expression1 Required. An expression that returns an IOAKAddln object.

expression2 Required. An expression that returns an IOAKAPI object.

result An OAKResult enumeration indicating the success or otherwise of the action.

Range Required Range. The range at which rows or columns are to be inserted.

copyFormula TRUE if the method is to fill the inserted cells with copies of their neighbours; FALSE if the inserted cells are to be left blank.

REMARKS

When applied to an IOAKAddln object, the Insert method performs the same action as the OAK Development | Insert Columns/Rows command. It inserts rows or columns at the current selection, even if that selection intersects array constants, array formulas or data tables. Just as the equivalent OAK user-interface command does, the command will ask the user whether or not to fill the inserted cells by copying the contents of neighboring cells.

When applied to an IOAKAPI object, the Insert method inserts rows or columns at the range indicated by the parameter submitted to the method. Whether or not to fill the inserted cells with copies of their neighbours is indicated by the second parameter to the method.

WRINKLES

To specify what is to be inserted unambiguously, Range needs to be one or more entire rows or one or more entire columns. It must consist of a single area only; a multi-area range will generate an error.
Though the Insert method can insert rows or columns that pass through array constants, array formulas or data tables, it cannot perform insertions that intersect pivot tables.

The InsertRowsOrColumns method will not work on worksheets that are protected.

5.24 IOAKAddIn

VBA API for OAK, using the user interface.

APPLIES TO

The Operis_OAK object library

ENUMERATIONS

OAKResult A code indicating the success or otherwise of an OAK user interface call.

METHODS

AnalyzeDiscrepancies : Activates the Analyze Discrepancies user interface.

ApplyNames : Activates the Apply Names user interface.

BuildNameDatabase : Activates the Build Name Database user interface.

CompareWorkbooks : Activates the Compare Workbooks user interface.

CompareWorksheets : Activates the Compare Worksheets user interface.

CompareRanges : Activates the Compare Ranges user interface.

CopyAddress : Activates the Copy Address user interface.

CopyLiteral : Activates the Copy Literal user interface.

DeapplyNames : Activates DeApply Names user interface.

Delete : Activates the Delete Rows/Columns user interface.

DeleteNamesInCells : Activates the Delete Names in Cells user interface.

GetVersion : Gets the current version of OAK.

Insert : Activates the Insert Rows/Columns user interface.

LocalizeNamesInCells : Activates the Localize Names in Cells user interface.

Map : Activates the Map Workbook user interface.
Optimize : Activates the Formula Optimize user interface.

Prune : Activates the Pruner user interface.

Reconstruct : Activates the Formula Reconstruct user interface.

RecreateNames : Activates the Recreate Names user interface.

RedefineName : Activates the Redefine Name user interface.

RemoveColorFormatting : Activates the Remove Color Formatting user interface.

RemoveHashREFNames : Activates the Remove #REF! Names user interface.

SearchFor : Activates the Search user interface.

SearchForArrays : Activates the Search for Arrays user interface.

SearchConstantFormula : Activates the Search for Constant Formulas user interface.

SearchForHardcodedConstant : Activates the Search for Hardcoded Constants user interface.

SearchForMerged : Activates the Search for Merged Cells user interface.

SearchForPrimaryError : Activates the Search for Primary Error Cells user interface.

SearchForReferenceToBlankCells : Activates the Search for References to Blank Cells user interface.

SearchForUnreferencedCells : Activates the Search for Unreferenced Cells user interface.

SearchForValue : Activates the Search for Unreferenced Cells user interface.

Summarize : Activates the Summarize Workbooks/Worksheets user interface.

Transpose : Activates the Transpose user interface.

WorksheetManager : Activates the Worksheet Manager.

UndoCompareModifications : Activates the Undo Compare Modifications user interface.

UnhideCells : Activates the Unhide Cells user interface.

PROPERTIES

ExcelApplication : The Excel application object. This must be set before any of the functions will work.
5.25  IOAKAPI

VBA API for OAK, bypassing the user interface.

APPLIES TO

The Operis_OAK object library

METHODS

- **AnalyzeDiscrepancies**: Produces a discrepancy analysis of the specified cells.
- **ApplyNames**: Applies the specified names to the specified range.
- **BuildNameDatabase**: Builds a database of the names in a workbook.
- **Compare**: Compare workbooks or worksheets, using the user's default colors.
- **CompareRanges**: Compare ranges, using the user's default colors.
- **CompareRangesWithColors**: Compare ranges, specifying colors.
- **CompareWithColors**: Compare workbooks or worksheets, specifying colors.
- **CopyLiteral**: Copies the formula(s) from the source range to the target range, preserving relative references.
- **DeapplyNames**: Deapplies the specified names to objects in the specified scope.
- **Delete**: Activates the Delete Rows/Columns user interface.
- **DeleteNamesInCells**: Activates the Delete Names in Cells user interface.
- **FormatAddress**: Formats the address of a specified range.
- **GetVersion**: Gets the current version of OAK.
- **Insert**: Activates the Insert Rows/Columns user interface.
- **LocalizeNamesInCells**: Localizes the names listed in the specified range.
- **Map**: Produces a map of the specified workbook or worksheets.
- **Optimize**: Activates the Optimizer user interface.
- **Prune**: Activates the Pruner user interface.
- **Reconstruct**: Reconstructs a range to a specified depth, and generates a report workbook.
Reformulate : Gives access to the technologies underpinning reconstruction, denaming and formula simplification.

RecreateNames : Renames adjacent blocks of cells to new names from text on an edge of the range, removing the existing names and remapping references to them.

RedefineName : Changes the name and/or value of the specified name, if necessary removing the existing names and remapping references to them.

RemoveColorFormatting : Removes the color formatting from the specified workbook or worksheets.

RemoveHashREFNames : Removes names whose definitions have acquired #REF! components.

SearchForArrays : Selects cells within the specified range that contain array formulas or array constants.

SearchConstantFormula : Selects cells within the specified range that contain constant formulas.

SearchForHardcodedConstant : Selects cells within the specified range that contain hardcoded (hardcoded) constants.

SearchForMerged : Selects cells within the specified range that contain merged cells.

SearchForPrimaryError : Selects cells within the specified range that are the origins of errors on the worksheet.

SearchForReferenceToBlankCells : Selects cells within the specified range that contain formulas that refer to blank cells.

SearchForUnreferencedCells : Selects cells within the specified range that are not referred to in any formulas.

SearchForValue : Selects cells within the specified range that contain cells with a range of values.

ShowProgress : Specify if user-interface progress indicators should be shown during API calls.

Summarize : Summarize a workbook or worksheets.

Transpose : Transposes the source range, optionally to the target range, while preserving addresses and removing name references.

UndoCompareModifications : Undoes any modifications made to the specified workbook or worksheets by previous OAK comparison actions.

UnhideCells : Exposes cells within the specified range that lie on hidden rows or columns.
PROPERTIES

ExcelApplication : The Excel application object. This must be set before any of the functions will work.

5.26 IRenameFailure

The type of the RemapFailures and RenameFailures properties of the IRenameResult interface which is returned by an invocation of IOAKAddIn.RedefineNames or IOAKAddIn.RecreateNames.

APPLIES TO
The Operis_OAK object library

PROPERTIES

Error A RenameErrorType describing what prevented a name from being changed.

name The name of the Excel name that has not been successfully changed.

5.27 IRenameResult

The return type of IOAKAddIn.RedefineNames or IOAKAddIn.RecreateNames

APPLIES TO
The Operis_OAK object library

PROPERTIES

RemapFailureCount Long

RemapFailures Variant An array of IRenameFailures

RenameFailureCount Long

RenameFailures Variant An array of IRenameFailures

Success Boolean

5.28 LocalizeNamesInCells

Turns into worksheet level names any workbook level names that happen to match the text in any one of the specified cells.
Example: You have workbook level names Dog and Cat. You highlight three cells that say Fish, Cat and Snake. `LocalizeNamesInCells` will cause the name Cat to be changed from a global (workbook level) name to a local (worksheet level) name, Sheet!Cat.

**APPLIES TO**

`IOAKAddIn`, `IOAKAPI`

**SYNTAX**

```
result1 = expression1 .LocalizeNamesInCells

expression2 .LocalizeNamesInCells(range1)
```

- `expression1` Required. An expression that returns an `IOAKAddIn` object.
- `expression2` Required. An expression that returns an `IOAKAPI` object.
- `result1` An OAKResult enumeration indicating the success or otherwise of the action.
- `range1` Required `Range`. The cells containing text, which will be used to identify names for localization.

**REMARKS**

When applied to an `IOAKAddIn` object, the `LocalizeNamesInCells` performs the same action as OAK's OAK Development | Names | Localize command, including presenting a confirmatory dialog box. The cells against which the global names are compared is the current selection, and the worksheet with which OAK associates the names is the active worksheet.

When applied to an `IOAKAPI` object, the `LocalizeNamesInCells` method

- the workbook level names that are altered are those that are defined in the workbook in which the cells specified by `range1` are located
- those names are compared with the contents of the cells specified as `range1`
- the worksheet with which OAK associates the names that match the cells is the worksheet on which `range1` is located.

This command is usually given immediately after use of Excel's `Create names` command, since that produces global names but offers no facilities for making them local.

When a name is turned from global to local, references to it on the same worksheet will continue to work, because local references to local names do not need to be qualified. References to a localized name on other worksheets do need to be qualified, from `=MyName` to `=Sheet!MyName`. OAK does not perform this step automatically. A clue that
such modification is necessary is that #NAME! errors appear in the worksheet after this method is activated.

### 5.29 Map

Generates a new workbook or worksheet which sets out high-level map of the cells in the active workbook or worksheet.

**APPLIES TO**

*IOAKAddIn*, *IOAKAPI*

**SYNTAX**

```plaintext
result1 = expression1 .Map

Set result2 = expression2 .Map(bookOrSheets, formulas, constants, constCopyValues, mapNamedRanges, mapArrays, outputTo, showLegend, colors)
```

- **expression1** Required. An expression that returns an *IOAKAddIn* object.
- **expression2** Required. An expression that returns an *IOAKAPI* object.
- **result1** An OAKResult enumeration indicating the success or otherwise of the action.
- **result2** The Workbook in which the worksheets generated or modified by the map action are to be found. Will be a new workbook, or an existing one, depending on the value of `outputTo`.
- **bookOrSheets** Required Variant, containing either a workbook that is, or a collection or array of Worksheets that are, to be mapped by OAK.
- **formulas** Required MapComponents. Indicates which kinds of formulas in the workbook are to be marked on the map generated by OAK: those resulting in number, logical, text or error values.
- **constants** Required MapComponents. Indicates which kinds of constants in the workbook are to be marked on the map generated by OAK: those resulting in number, logical, text or error values.
- **constCopyValues** Required Boolean. Indicates how constant values are to be shown on the map. *True* if the constants are themselves to be shown on the map. *False* if numeric values are indicated by #; text values by X, logical values by L, and error values by Err.
- **mapNamedRanges** Required Boolean. True is named ranges in the workbook or worksheet(s) are to be marked on the map generated by OAK. False if named ranges are not to be marked.
mapArrays Required Boolean. True if array formulas or constants in the workbook or worksheet(s) are to be marked on the map generated by OAK. False if arrays are not to be marked.

outputTo Required MapOutput. Indicates whether the map generated by OAK is to be presented in a new workbook, as new worksheets in the workbook being mapped, or is to be overlayed on the worksheets being mapped.

showLegend Required Boolean. True if OAK is to add alongside the worksheets bearing the generated maps an additional worksheet bearing a legend indicating the notation being used. False if the legend worksheet is not to be generated.

colors Optional Scripting.Dictionary of MapColor to long, specifying a map item and its color.

REMARKS

When applied to an IOAKAddIn object, the Map method activates the same dialog box as is presented when the Map Workbook/Worksheets command is selected manually from the OAK user interface. The action that follows is controlled by the data gathered by the dialog box from the user, and acts on the workbook or worksheet(s) specified in that dialog.

When applied to an IOAKAPI object, the Map method generates a map identical to the one that would be generated by OAK if its Map Workbook/Worksheets command were activated manually from the user interface, but the details of the analysis, including the workbook or worksheet(s) to be mapped, are specified by the parameters submitted to the method.

OAK provides the means to change the color scheme used in the maps it generates. The command that does this, OAK Development/Review | Options | Colors | Set map colors, is offered through the user interface. The function is not accessible through the OAK API, but the Map method will use any colors specified by item. The Map method can override these colors by supplying a Scripting.Dictionary for the optional colors parameter.

EXAMPLE

To generate the most complete map (showing named ranges, arrays and every kind of formula and constant) of the active workbook:

```vba
Set MapWorkbook = MyOakApi.Map(ActiveWorkbook, MapComponents_All, MapComponents_All, True, True, True, MapOutput_NewWorkbook, True)
```

To generate the same map, but overriding some of the colors:

```vba
Dim colors As Object
Set colors = CreateObject("Scripting.Dictionary")
```
' Override some of the map colors
colors.Add MapColor_Array, Win32Color(255, 0, 0) ' red arrays
colors.Add MapColor_NamedRange, Win32Color(0, 255, 0) ' green named ranges

Set MapWorkbook = MyOakApi.Map(ActiveWorkbook, MapComponents_All, MapComponents_All, True, True, True, MapOutput_NewWorkbook, True, colors)

For the Win32Color function, see the section titled: Colors in compare and map commands

5.30 MapComponents

A set of constants that specifies to OAK’s Map method which kinds of formulas or constants in the workbook are to be marked on a Map generated by OAK: those resulting in number, logical, text or error values.

APPLIES TO
Operis_OAK

SYNTAX

The two parameters in the Map method that specify the kind of values that are to be included on a map generated by OAK can be

1 MapComponents_All
2 MapComponents_None
3 The sum of any combination of
   - MapComponents_Errors
   - MapComponents_Numbers
   - MapComponents_Logicals
   - MapComponents_Text

5.31 MapColor

A set of constants that in combination with color values, specifies to OAK’s Map method which map kinds of formulas or constants in the workbook are to be marked on a Map generated by OAK: those resulting in number, logical, text or error values.

APPLIES TO
Operis_OAK
SYNTAX

The two parameters in the Map method that specify the kind of values that are to be included on a map generated by OAK can be

- MapColor_DistinctFormulaForeground
- MapColor_DistinctFormulaBackground
- MapColor_TopConsistentFormulaForeground
- MapColor_TopConsistentFormulaBackground
- MapColor_LeftConsistentFormulaForeground
- MapColor_LeftConsistentFormulaBackground
- MapColor_TopLeftConsistentFormulaForeground
- MapColor_TopLeftConsistentFormulaBackground
- MapColor_TextConstantForeground
- MapColor_TextConstantBackground
- MapColor_LogicalConstantForeground
- MapColor_LogicalConstantBackground
- MapColor_NumberConstantForeground
- MapColor_NumberConstantBackground
- MapColor_ErrorCellForeground
- MapColor_ErrorCellBackground
- MapColor_NamedRange
- MapColor_PrintAreaPrintTitle
- MapColor_Array

5.32 MapOutput

A set of constants that specifies to OAK's Map method where a map is to be generated.

APPLIES TO
Operis_OAK

SYNTAX

The parameter in the Map method that specifies the where a map is to be generated can be

- MapOutput_NewWorkbook, if the map is to be presented in a newly created workbook
- MapOutput_Overlay, if the map is to be overlaid on the workbook or worksheet(s) being mapped
- MapOutput_SameWorkbook, if the map is to be presented as worksheet(s) newly added to the workbook being mapped

5.33 OAKResult

An enumeration returned by a call to the IOAKAddIn interface which indicates whether the call was successful or not.
Possible values are

- **OAKResult_Finished**: The call executed successfully.
- **OAKResult_Cancelled**: The user interface was presented to the user as a result of the IOAKAPI call but the user abandoned the operation by clicking on the user interface's Cancel button or pressing Escape.
- **OAKResult_Aborted**: The user interface was presented to the user as a result of the IOAKAPI call and the user activated it by pressing the user interface's OK key or pressing Return, but abandoned the action as it was happening by clicking on the Cancel button on the progress indicator or pressing Escape.
- **OAKResult_InputError**: The user interface was presented to the user as a result of the IOAKAPI call but no action was taken because the inputs provided by the user were unsuitable.
- **OAKResult_Exception**: The user interface was presented to the user, and populated with suitable inputs, but the attempt to act on them failed.
- **OAKResult_Reentrant**: OAK was already doing something when the function was called. This is intended for the single document interface (SDI) of Excel 2013-2016, where an OAK function could be activated from one document's ribbon while another that has been initiated from another document's ribbon is still running.

**APPLIES TO**

Most of the methods of IOAKAddIn.

**EXAMPLE**

```vba
Sub Main()
    Dim o As Operis_OAK.IOAKAddIn
    Dim r As Operis_OAK.OAKResult
    Set o = CreateObject("Operis.OAK.Connect")
    Set o.ExcelApplication = Application
    r = o.BuildNameDatabase
    Select Case r
        Case OAKResult_Finished
            MsgBox "Build name database completed successfully"
        Case OAKResult_Cancelled
            MsgBox "Build name database was cancelled by user"
        Case OAKResult_InputError
            MsgBox "Build name database action not attempted: inputs provided by user were unsatisfactory"
        Case OAKResult_Exception
            MsgBox "Build name database action attempted but failed"
        Case OAKResult_Reentrant
            MsgBox "Build name database action attempted but failed because OAK was busy"
        Case Else
            MsgBox "Build name database action returned a value that it should never return"
    End Select
```

5.34 Operis_OAK

The object library by which OAK exposes its interface to programs written in VBA or similar languages.

APPLIES TO

IOAKAddIn, IOAKAPI

MEMBERS

IOAKAddIn: Interface to the methods by which the OAK user interface can be activated.

IOAKAPI: Interface to the methods by which OAK can be driven programmatically.

ICompareResult: Interface providing details of a workbook or worksheet comparison

IDeapplyNamesResult: Interface providing details of the outcome of an attempt to deapply names

IRenameFailure: Interface providing details of where an attempt to rename names proved unsuccessful

IRenameResult: Interface for providing details of the outcome of an attempt to rename names

FormatAddressDelimiter: Enumeration containing constants for specifying how to format the output of the FormatAddress command.

MapComponents: Enumeration containing constants for specifying what kinds of formulas and constants are to be shown on a worksheet map generated by OAK

MapOutput: Enumeration containing constants for specifying where a worksheet map generated by OAK is to be placed.

5.35 Optimize

Applies a number of transformations to a formula that aim to remove redundancy and complexity.

APPLIES TO

IOAKAddIn, IOAKAPI
SYNTAX

result1 = expression1 .Optimize

expression2 .Optimize(range1, mode)

expression1  Required. An expression that returns an IOAKAddIn object.
expression2  Required. An expression that returns an IOAKAPI object.
result1  An OAKResult enumeration indicating the success or otherwise of the action.
range1  Required Range. The range in which to which to apply the transformations,
mode  Required OptimizationMode: One of OptimizationMode_Aggressive, OptimizationMode_Conservative, or OptimizationMode_None

REMARKS

When applied to an IOAKAddIn object, the Optimize method acts on the current selection to perform the same action as the OAK Development | Formula | Optimize command, including presentation of a dialog box which asks whether to apply aggressive or conservative transformations, and another to confirm the replacement of the untransformed formula by the transformed one.

When applied to an IOAKAPI object, the Optimize method acts on the cells indicated by Range1, and is told whether to apply aggressive or conservative transformations by the parameter mode.

WRINKLES

When applied to an IOAKAddIn object, the Optimize method follows the One-cell rule.
When applied to an IOAKAPI object, it does not: if supplied a Range parameter describing a single cell, it will look in that cell only.

When applied to an IOAKAddIn object, the method will display a dialog box if no useful change can be made to any formula, just as the equivalent OAK user-interface command does.

It makes little sense to pass this method a mode of OptimizationMode_None. It won't do anything. The option is provided because the same parameter is also passed into the methods Reconstruct and Reformulate, where it is a legitimate choice.

5.36 Prune

Simplifies formulas in cells by removing branches of conditional or lookup functions that are inactive given current inputs.
APPLIES TO

`IOAKAddIn` , `IOAKAPI`

SYNTAX

```plaintext
result1 = expression1 .Prune

expression2 .Prune(Range1, mode, sumExpansionLimit, preserveNames)
```

- `expression1` Required. An expression that returns an `IOAKAddIn` object.
- `expression2` Required. An expression that returns an `IOAKAPI` object.
- `result1` An OAKResult enumeration indicating the success or otherwise of the action.
- `Range1` Required `Range`. The range in which to search for cells that are not mentioned by any formula.
- `mode` Required `PruningMode`. Indicates which functions are to be pruned by the method.
- `sumExpansionLimit` Required `Integer`. Any SUM() function which acts on this number or fewer cells will be expanded, that is, turned from SUM(A1:A3) to A1+A2+A3.
- `preserveNames` Required `Boolean`. `True` if the pruned formula is to reuse any names that were present in the original. `False` if names are to be removed from the formula.

REMARKS

When applied to an `IOAKAddIn` object, the `Prune` method performs the same action as the OAK Review | Formula | Prune command, including presenting the same dialog boxes. It acts on the current selection.

When applied to an `IOAKAPI` object, the `Prune` method acts on the cells specified by `Range1`.

WRINKLES

When applied to an `IOAKAddIn` object, the `Prune` method does not follow the One-cell rule. This departure from the practice of Excel and OAK in other functions is intentional. Pruning is a potentially destructive action, delivering a spreadsheet that gives unchanged answers with current inputs, but which might give quite wrong answers with other inputs. It is only permitted to act on cells that are selected explicitly.
When applied to an \texttt{IOAKAddIn} object, the method will display a dialog box before performing the pruning, to ask which functions are to be pruned, just as the equivalent OAK user-interface command does.

It is for any code to ensure that worksheets are thoroughly recalculated before using these methods. It is for the user to keep a copy of the spreadsheet before letting OAK alter it through pruning.

### 5.37 PruningMode

A set of constants that specify, to methods where it is relevant, which functions are to be pruned.

**APPLIES TO**

\textit{Operis\_OAK}

**SYNTAX**

Where the methods AnalyzeDiscrepancies, Prune, Reconstruct, or Reformulate need to be provided with a parameter that specifies the kind of expression pruning that is to be performed, they can be given

1. PruningMode\_Everything
2. PruningMode\_None
3. The sum of any combination of
   - PruningMode\_AVERAGEIF
   - PruningMode\_AVERAGEIFS
   - PruningMode\_CHOOSE
   - PruningMode\_COLUMN
   - PruningMode\_COUNTIF
   - PruningMode\_COUNTIFS
   - PruningMode\_HLOOKUP
   - PruningMode\_IF
   - PruningMode\_INDEX
   - PruningMode\_INDIRECT
   - PruningMode\_LOOKUP
   - PruningMode\_MATCH
Reference material: OAK object reference

- PruningMode_MAX
- PruningMode_MIN
- PruningMode_OFFSET
- PruningMode_ROW
- PruningMode_SUMIF
- PruningMode_SUMIFS
- PruningMode_VLOOKUP

EXAMPLE

To analyse the differences in derivation between two cells MyRange1 and MyRange2, back to 4 levels of precedence, pruning occurrences of the COUNTIF and SUMIF functions

```
MyOakApi.AnalyzeDiscrepancies MyRange1, MyRange2, 4, False, 0, True, 500,
PruningMode_COUNTIF + PruningMode_SUMIF
```

5.38 Reconstruct

Generates a new worksheet which seeks to make a calculation understandable by laying out its essential features simply.

APPLIES TO

- IOAKAddIn
- IOAKAPI

SYNTAX

```
result = expression1 .Reconstruct

expression2 .Reconstruct(Range1, OptimizationMode1, PruningMode1, 
additionsOnly, sumExpansionLimit, maxDepth)
```

- `expression1` Required. An expression that returns an `IOAKAddIn` object.
- `expression2` Required. An expression that returns an `IOAKAPI` object.
- `result` An OAKResult enumeration indicating the success or otherwise of the action.
- `Range1` Required `Range`. The cells containing the calculation to be reconstructed.

- `OptimizationMode1` Required. `OptimizationMode`: One of
  - OptimizationMode_Aggressive
  - OptimizationMode_Conservative
  - OptimizationMode_None
PruningModel Required PruningMode. Specifies what, if any, pruning is performed on functions encountered during the analysis of precedent formulas.

additionsOnly Required Boolean. Instructs OAK to perform precedent substitution only to the extent that the result is a formula that contains +, - and SUM. Equivalent to the Confine to additions option in the dialog box presented when the user interface version of the Reconstruct command is activated.

sumExpansionLimit Required Long. Instructs OAK to expand any SUM() function which acts on this number or fewer cells that is, convert from SUM(A1:A3) to A1+A2+A3.

maxDepth Required Long. Indicates how many levels of precedent substitution are to be attempted.

REMARKS

When applied to an IOAKAddIn object, the Reconstruct method performs the same action as the OAK Review | Formula | Reconstruct command, including presenting the same dialog boxes.

When applied to an IOAKAPI object, the Reconstruct method acts on the cells specified by Range1, delivering a reconstruction specified by the other parameters to the method.

WRINKLES

When applied to an IOAKAddIn object, the Reconstruct method does not follow the One-cell rule. This is because it acts on a calculation, a notion defined as neighboring cells that form all or part of a single row and contain the same formula. The method will display a dialog box seeking details of the action it is to perform, just as the equivalent OAK user-interface command does.

Though OAK will attempt the specified number of levels of precedent substitution, it may deliver fewer of them.

5.39 Reformulate

The Reformulate method is the only one offered by IOAKAPI which does not correspond with a command offered by the OAK user interface.

OAK’s facilities to optimize, prune, dename and reconstruct a formula all involve parsing (that is, interpreting) the expression involved, applying some transformations, and writing out the revised expression. In a reconstruction, the revised expression is written on a new workbook, but in all other cases, the expression is written back to the original formula location. So a formula dnamed by OAK Development | Names | Deapply, or simplified algebraically by OAK Development | Formula | Optimize, can be thought of as an in-place reconstruction of that formula.
The Reformulate method exposes this mechanism for general use. With the appropriate choice of parameters, many of OAK's optimizing, denaming, pruning and reconstruction methods can be synthesized from it. The method is offered for use by experts who may wish to use other combinations of parameters to build up their own applications of this technology for manipulating and gaining insight into spreadsheet calculations.

**APPLIES TO**

**IOAKAPI**

**SYNTAX**

```
expression = Reformulate( Range1, DelinkMode1, OptimizationMode1, PruningMode1, additionsOnly, sumExpansionLimit, maxDepth )
```

expression  Required. An expression that returns an **IOAKAPI** object.

**Range1**  Required **Range**. The cells containing the calculation to be reconstructed.

**DelinkMode1**  Required **DelinkMode**. One of **DelinkMode_ConstantFormula**, **DelinkMode_Constants**, **DelinkMode_Everything**, **DelinkMode_Formula**, **DelinkMode_NonConstantFormula**, **DelinkModeNone**. Delinking is what distinguishes an Optimized expression, written back to its original location and referring to its original precedents, from a Reconstruction, in which a formula is written to a new location, and is altered so that it refers to its precedents indirectly through new cells. In brief, it replaces a reference with what it refers to. For example, a cell containing the formula =B5, where B5 contains the formula =C7, would be repopulated with the formula =C7.

**OptimizationMode1**  Required **OptimizationMode**: One of **OptimizationMode_Aggressive** or **OptimizationMode_Conservative**

**PruningMode1**  Required **PruningMode**. Specifies what, if any, pruning is performed on functions encountered during the analysis of precedent formulas.

**additionsOnly**  Required **Boolean**. Instructs OAK to perform precedent substitution only to the extent that the result is a formula that contains +, - and SUM. Equivalent to the Confine to additions option in the dialog box presented when the user interface version of the Reconstruct command is activated.

**sumExpansionLimit**  Required **Long**. Instructs OAK to expand any SUM() function which acts on this number or fewer cells that is, convert from SUM(A1:A3) to A1+A2+A3.

**maxDepth**  Required **Long**. Indicates how many levels of precedent substitution are to be attempted.
REMARKS

Since it has no analog in the OAK user interface, no version of this method is offered for the `IOAKAddIn` object.

5.40 RecreateNames

Deletes a specified name, creates a new one, and changes all formulas and names that refer to the old name so that they refer instead to the new one.

APPLIES TO

`IOAKAddIn`, `IOAKAPI`

SYNTAX

Set `result1 = expression1`.RecreateNames

Set `result2 = expression2`.RecreateNames(Range1, labelRegion, forceLocal, [scope])

-expression1  Required. An expression that returns an `IOAKAddIn` object.
-expression2  Required. An expression that returns an `IOAKAPI` object.
-result1  An `OAKResult` enumeration indicating the success or otherwise of the action.
-result2  An `IRenameResult` which indicates the success or otherwise of the action.
-range1  A `Range` containing the cells to be renamed, with their new names along the edges.

=labelRegion  Required `NamesSelection`, indicating from which edge of `range1` the new names are to be taken. One of `NamesSelection_Bottom`, `NamesSelection_Left`, `NamesSelection_None`, `NamesSelection_Right`, `NamesSelection_Top`.

-forceLocal  Required `Boolean`. If `True`, the names created will be local, associated with the worksheet on which they are created. If `False`, the names created will be local only if they being with an exclamation mark. Otherwise, they will be global, associated with the workbook in which it is created.

-scope  Optional. One of

- an Excel `Workbook` object, if OAK is to adjust any formulas which refer to the names being replaced in that workbook.
- an Excel `Worksheet` object, if OAK is to adjust any formulas which refer to the names being replaced in that worksheet.
**Nothing**, if OAK is to adjust any formulas which refer to the names being replaced in any open workbook.

**REMARKS**

When applied to an `IOAKAddIn` object, the **RecreateNames** method activates the same dialog box as is presented when the **Recreate names** command is selected manually from the OAK user interface. The action that follows is controlled by the data gathered by the dialog box from the user.

When applied to an `IOAKAPI` object, the **RecreateNames** method changes names as specified by the parameters submitted to the method.

### 5.41 **RedefineName**

Deletes a specified name, creates a new one, and changes all formulas and names that refer to the old name so that they refer instead to the new one.

**APPLIES TO**

`IOAKAddIn`, `IOAKAPI`

**SYNTAX**

```plaintext
Set result1 = expression1 .RedefineName

Set result2 = expression2 .RedefineName(name1, name2, scope)
```

- `expression1` Required. An expression that returns an `IOAKAddIn` object.
- `expression2` Required. An expression that returns an `IOAKAPI` object.
- `result1` An `OAKResult` enumeration indicating the success or otherwise of the action.
- `result2` An `IRenameResult` which indicates the success or otherwise of the action.
- `name1` Required Name. The name that is to be changed.
- `name2` Required String. The new name that is to be used.
- `scope` Optional. One of
  - an Excel **Workbook** object, if OAK is to adjust any formulas which refer to the names being replaced in that workbook.
  - an Excel **Worksheet** object, if OAK is to adjust any formulas which refer to the names being replaced in that worksheet.
• **Nothing**, if OAK is to adjust any formulas which refer to the names being replaced in any open workbook.

**REMARKS**

When applied to an `IOAKAddIn` object, the RedefineName method activates the same dialog box as is presented when the Redefine name command is selected manually from the OAK user interface. The action that follows is controlled by the data gathered by the dialog box from the user.

When applied to an `IOAKAPI` object, the RedefineName method changes names as specified by the parameters submitted to the method.

### 5.42 RemoveColorFormatting

Removes potentially distracting color formatting from specified workbook or worksheets.

**SYNTAX**

```expression1 .RemoveColorFormatting```

```expression2 .RemoveColorFormatting(BookOrSheets )```

- **expression1** Required. An expression that returns an `IOAKAddIn` object.
- **expression2** Required. An expression that returns an `IOAKAPI` object.

**BookOrSheets** Required. One of

- an Excel workbook object
- an Excel worksheet object
- a collection of worksheet objects
- an array of worksheet objects

**REMARKS**

When applied to an `IOAKAddIn` object, the Remove color formatting method activates the same dialog box as is presented when the Remove color formatting command is selected manually from the OAK user interface. The action that follows is controlled by the data gathered by the dialog box from the user.

When applied to an `IOAKAPI` object, the Remove color formatting removes formatting from the workbook or worksheet or worksheets specified by the parameters submitted to the method.
EXAMPLE

To remove color formatting from the active worksheet:

    MyOakApi.RemoveColorFormatting ActiveWorksheet

5.43  RemoveHashRefNames

Removes any names that have #REF! in their definitions, a sign that they probably refer to cells that have been deleted.

APPLIES TO

IOAKAddIn

SYNTAX

    result1 = expression1 .RemoveRefErrorNames

expression1   Required. An expression that returns an IOAKAddIn object.
result1       An OAKResult enumeration indicating the success or otherwise of the action.

REMARKS

The RemoveRefErrorNames method performs the same action as the OAK Development | Names | Remove #REF! command.

The command deletes names that contain "#REF!" anywhere in the definition. This is usually a sign that the cells to which the names relate have been deleted; if the cells are gone, the name might as well go too. But it isn't always so. The formula =IF(ROW(A1) =1,#REF!,23) is a legitimate formula, but OAK will delete it anyway because it contains the unwanted string. For this reason OAK presents a confirmatory dialog before doing the deletion.

Since there is no neat way of interacting programmatically with that dialog, no version of RemoveRefErrorNames is provided for the IOAKAPI method.

5.44  RenameErrorType

An enumeration used to specify an error returned in an IRenameFailure object returned in an IRenameResult object by an invocation of IOAKAddIn.RedefineNames or IOAKAddIn.RecreateNames

APPLIES TO

Operis_OAK
SYNTAX

- RenameErrorType_None
- RenameErrorType_ExcelError
- RenameErrorType_NameRepeated
- RenameErrorType_ClashesWithExisting
- RenameErrorType_Localization
- RenameErrorType_Globalization
- RenameErrorType_MixedScope
- RenameErrorType_Invalid

5.45 SearchFor

Searches a specified area of a worksheet for cells that have been merged, selecting any that are found in a multi-area range.

APPLIES TO

IOAKAddIn, IOAKAPI

SYNTAX

result1 = expression1 .SearchFor

Set result2 = expression2 .SearchFor(Range1, Text, SearchType, MatchCase)

expression1 Required. An expression that returns an IOAKAddIn object.
expression2 Required. An expression that returns an IOAKAPI object.
result1 An OAKResult enumeration indicating the success or otherwise of the action.
result2 A Range, set either to all the merged cells found within Range1, or to Nothing if no merged cells are present.

Range1 Required Range. The range in which to search for the specified text.
Text Required String. The text to be searched for.

SearchType SearchContentType. Indicates what kind of search to perform. One of SearchMatchType_Contains, SearchMatchType_DoesNotContain, SearchMatchType_EndsWith, SearchMatchType_Equals, SearchMatchType_GreaterThan, SearchMatchType_GreaterThanOrEqualTo, SearchMatchType_IsBetween, SearchMatchType_IsNotBetween, SearchMatchType_LessThan, SearchMatchType_LessThanOrEqual, SearchMatchType_NotEquals, SearchMatchType_StartsWith.

MatchCase Required Boolean. True if the search is to be case sensitive. False if the search is to ignore any difference between the casing of the specified text and the contents of the cells being searched.

REMARKS

When applied to an IOAKAddIn object, the SearchFor method performs some of the same action as the OAK Development/Review | Search | Conditional search command. It search for cells within the current selection that satisfy the required criteria. Other actions performed by that command are implemented by the SearchForValue method.

When applied to an IOAKAPI object, the SearchFor method performs a conditional search within the range indicated by the parameter submitted to the method.

WRINKLES

When applied to an IOAKAddIn object, the SearchFor method follows the One-cell rule. When applied to an IOAKAPI object, it does not: if supplied a Range parameter describing a single cell, it will look in that cell only.

When applied to an IOAKAddIn object, the method will display a dialog box if zero or one area of merged cells is detected, just as the equivalent OAK user-interface command does.

EXAMPLE

To highlight those cells, if any, in the current selection that make use of the deprecated function OFFSET in their formula:

```vbnet
Dim r as Range
Set r = MyOakApi.SearchFor(Selection, "OFFSET(", SearchContentType_Formula, True)
If Not r Is Nothing then r.Select
```

5.46 SearchForArrays

Searches a specified area of a worksheet for cells array contants or formulas, selecting any that are found in a multi-area range.
APPLIES TO

IOAKAddin, IOAKAPI

SYNTAX

\[
\text{result1} = \text{expression1}\cdot\text{SearchForArrays}
\]

Set \[
\text{result2} = \text{expression2}\cdot\text{SearchForArrays}\left(\text{Range1}\right)
\]

- **expression1**: Required. An expression that returns an `IOAKAddin` object.
- **expression2**: Required. An expression that returns an `IOAKAPI` object.
- **result1**: An OAKResult enumeration indicating the success or otherwise of the action.
- **result2**: A `Range` consisting of all the arrays found within `Range1`, or Nothing if no arrays are found.
- **Range1**: Required Range. The range in which to search for arrays.

REMARKS

When applied to an `IOAKAddin` object, the `SearchForArrays` method performs the same action as the OAK Development/Review | Search | Arrays command. It searches for arrays within the current selection.

When applied to an `IOAKAPI` object, the `SearchForArrays` method applies searches for arrays within the the range indicated by the parameter submitted to the method.

WRINKLES

When applied to an `IOAKAddin` object, the `SearchForArrays` method follows the One-cell rule. When applied to an `IOAKAPI` object, it does not: if supplied a Range parameter describing a single cell, it will look in that cell only.

When applied to an `IOAKAddin` object, the method will display a dialog box if no arrays are detected, just as the equivalent OAK user-interface command does.

The OAK method finds all the arrays in the selection. The Goto | Special | Current array method built in to Excel can only find one array at a time.

5.47 **SearchForConstantFormula**

Searches a specified area of a worksheet for cells that have contain a constant formula, selecting any that are found in a multi-area range.
An example of a constant formula is \(\frac{22}{7}\). It is a formula, in that it starts with an equals sign; but it makes no reference to other cells as precedents, so it will always give the same answer. The danger with such cells is that they might be overlooked when performing an exhaustive check of the inputs to a spreadsheet, since they will not appear in a listing of constants produced by OAK or any similar package, nor be identified by Home | Find & Select | Go To Special | Constants command.

APPLIES TO

\texttt{IOAKAddIn, IOAKAPI}

SYNTAX

\[
\text{result1} = \text{expression1} . \text{SearchForConstantFormula}
\]

\[
\text{Set result2} = \text{expression2} . \text{SearchForConstantFormula(Range1)}
\]

expression1 Required. An expression that returns an \texttt{IOAKAddIn} object.

expression2 Required. An expression that returns an \texttt{IOAKAPI} object.

result1 An OAKResult enumeration indicating the success or otherwise of the action.

result2 A \texttt{Range} consisting of all the cells containing a constant formula found within Range1, or Nothing if no such cells are found

Range1 Required \texttt{Range}. The range in which to search for cells containing a constant formula.

REMARKS

When applied to an \texttt{IOAKAddIn} object, the \texttt{SearchForConstantFormula} method performs the same action as the OAK Development | Search | Constant formula cells command. It searches for cells containing a constant formula within the current selection.

When applied to an \texttt{IOAKAPI} object, the \texttt{SearchForConstantFormula} method applies searches for cells containing a constant formula within the range indicated by the parameter submitted to the method.

WRINKLES

When applied to an \texttt{IOAKAddIn} object, the \texttt{SearchForConstantFormula} method follows the One-cell rule. When applied to an \texttt{IOAKAPI} object, it does not: if supplied a Range parameter describing a single cell, it will look in that cell only.
When applied to an `IOAKAddIn` object, the method will display a dialog box if no cells containing a constant formula are detected, just as the equivalent OAK user-interface command does.

5.48 SearchForHardcodedConstant

Searches a specified area of a worksheet for cells that have contain a hardcoded constant, selecting any that are found in a multi-area range.

An example of a hardcoded constant is =A5*365. The 365 is an assumption, presumably the number of days in year. The danger with embedding such assumptions in a formula is that they might be overlooked when performing an exhaustive check of the inputs to a spreadsheet, since they will not appear in a listing of constants produced by OAK or any similar package, nor be identified by Excel's Home | Find & Select | Go To Special | Constants command.

APPLIES TO

`IOAKAddIn`, `IOAKAPI`

SYNTAX

```
result1 = expression1 .SearchForHardcodedConstant
Set result2 = expression2 .SearchForHardcodedConstant(Range1)
```

- `expression1` Required. An expression that returns an `IOAKAddIn` object.
- `expression2` Required. An expression that returns an `IOAKAPI` object.
- `result1` An OAKResult enumeration indicating the success or otherwise of the action.
- `result2` A `Range` consisting of all the cells found within `Range1` that contain hardcoded constants, or Nothing if no such cells are found.
- `Range1` Required `Range`. The range in which to search for cells that contain hardcoded constants.

REMARKS

When applied to an `IOAKAddIn` object, the `SearchForHardcodedConstant` method performs the same action as the OAK Development | Search | Hardcoded constants command. It searches for cells containing a hardcoded constant within the current selection.
When applied to an `IOAKAPI` object, the `SearchForHardcodedConstant` method applies searches for cells containing a hardcoded constant within the range indicated by the parameter submitted to the method.

WRINKLES

When applied to an `IOAKAddIn` object, the `SearchForHardcodedConstant` method follows the One-cell rule. When applied to an `IOAKAPI` object, it does not: if supplied a Range parameter describing a single cell, it will look in that cell only.

When applied to an `IOAKAddIn` object, the method will display a dialog box if no cells containing a hardcoded constant are detected, just as the equivalent OAK user-interface command does.

5.49 SearchForMerged

Searches a specified area of a worksheet for cells that have been merged, selecting any that are found in a multi-area range.

APPLIES TO

`IOAKAddIn`, `IOAKAPI`

SYNTAX

```
result1 = expression1 .SearchForMerged

Set result2 = expression2 .SearchForMerged(Range1)
```

- `expression1` Required. An expression that returns an `IOAKAddIn` object.
- `expression2` Required. An expression that returns an `IOAKAPI` object.
- `result1` An OAKResult enumeration indicating the success or otherwise of the action.
- `result2` Range containing of all the merged cells found within `Range1`, or Nothing if no such cells are found.
- `Range1` Required `Range`. The range in which to search for merged cells.

REMARKS

When applied to an `IOAKAddIn` object, the `SearchForMerged` method performs the same action as the OAK Development | Search | Merged cells command. It searches for merged cells within the current selection.
When applied to an `IOAKAPI` object, the `SearchForMerged` method applies searches for merged cells within the range indicated by the parameter submitted to the method.

WRINKLES

When applied to an `IOAKAddIn` object, the `SearchForMerged` method follows the One-cell rule. When applied to an `IOAKAPI` object, it does not: if supplied a Range parameter describing a single cell, it will look in that cell only.

When applied to an `IOAKAddIn` object, the method will display a dialog box if zero or one area of merged cells is detected, just as the equivalent OAK user-interface command does.

EXAMPLE

This code fragment identifies any merged cells in the active worksheet and replaces the merge with the more desirable Center across selection formatting.

```vba
Option Explicit

Sub Main()
    Dim o As Operis_OAK.IOAKAPI, r As Range
    Set o = CreateObject("Operis.OAK.Connect")
    Set o.ExcelApplication = Application
    Set r = o.SearchForMerged(ActiveSheet.Cells)
    If r is Nothing then Exit Sub
    r.UnMerge
    r.HorizontalAlignment = xlCenterAcrossSelection
End Sub
```

5.50 `SearchForPrimaryError`

Searches a specified area of a worksheet for cells that are the primary causes of error values, selecting any that are found in a multi-area range.

A primary error cell is one that results in an error value even though none of its precedents are supplying error values to it. It is an origin of error values, rather than merely propagating through the spreadsheets errors that have arisen elsewhere.

APPLIES TO

`IOAKAddln`, `IOAKAPI`

SYNTAX

```vba
result1 = expression1 .SearchForPrimaryError
```
Set result2 = expression2 .SearchForPrimaryError(Range1)

expression1 Required. An expression that returns an IOAKAddIn object.
expression2 Required. An expression that returns an IOAKAPI object.
result1 An OAKResult enumeration indicating the success or otherwise of the action.
result2 A Range containing of all the primary error cells found within Range1, or Nothing if no such cells are found.

Range1 Required Range. The range in which to search for primary error cells.

REMARKS
When applied to an IOAKAddIn object, the SearchForPrimaryError method performs the same action as the OAK Development | Search | Primary error cells command. It searches for primary error cells within the current selection.

When applied to an IOAKAPI object, the SearchForPrimaryError method applies searches for primary error cells within the the range indicated by the parameter submitted to the method.

WRINKLES
When applied to an IOAKAddIn object, the SearchForPrimaryError method follows the One-cell rule. When applied to an IOAKAPI object, it does not: if supplied a Range parameter describing a single cell, it will look in that cell only.

When applied to an IOAKAddIn object, the method will display a dialog box if no primary error cells are detected, just as the equivalent OAK user-interface command does.

If you select a small number of cells on a worksheet and then select one of the SearchForPrimaryError methods, the resulting cells might not actually be primary error cells. This is because they might refer to error cells outside your selection. It may be best to run this command on a whole worksheet.

However, you may find that even then, the cells which are found by the command do not constitute primary errors because they refer to error cells on a different worksheet in your workbook. In this case you will have to run the methods on that worksheet to find the true primary error cells.

5.51 SearchForReferencesToBlankCells

Searches a specified area of a worksheet for cells that refer to other cells that are blank, selecting any that are found in a multi-area range.
A spreadsheet user might feel that if a cell is blank, he is free to type something into it. A formula that refers to that blank cell could then start giving an unintended result. In some cases, a reference to a blank cell is simply the result of mistyping the address of a cell that is not blank.

APPLIES TO

\textit{IOAKAddIn}, \textit{IOAKAPI}

SYNTAX

\begin{verbatim}
result1 = expression1 .SearchForReferencesToBlankCells
Set result2 = expression2 .SearchForReferencesToBlankCells(Range1)
\end{verbatim}

\textit{expression1} Required. An expression that returns an \textit{IOAKAddIn} object.

\textit{expression2} Required. An expression that returns an \textit{IOAKAPI} object.

\textit{result1} An OAKResult enumeration indicating the success or otherwise of the action.

\textit{result2} A Range containing of all the cells found within \textit{Range1} that refer to other cells that are blank, or \textbf{Nothing} if no such cells are found.

\textit{Range1} Required \textbf{Range}. The range in which to search for cells containing a formula that refers to other cells that are blank.

REMARKS

When applied to an \textit{IOAKAddIn} object, the \textbf{SearchForReferencesToBlankCells} method performs the same action as the OAK Development/Review \textquoteleft Search \textquoteleft References to Blank Cells command. It searches for cells within the current selection that refer to blank cells.

When applied to an \textit{IOAKAPI} object, the \textbf{SearchForReferencesToBlankCells} method applies searches for cells within the the range indicated by the parameter submitted to the method that refer to blank cells.

WRINKLES

When applied to an \textit{IOAKAddIn} object, the \textbf{SearchForReferencesToBlankCells} method follows the One-cell rule. When applied to an \textit{IOAKAPI} object, it does not: if supplied a Range parameter describing a single cell, it will look in that cell only.

When applied to an \textit{IOAKAddIn} object, the method will display a dialog box if no references to blank cells are detected, just as the equivalent OAK user-interface command does.
Neither method will detect a reference to a blank cell that is located on a different worksheet from the formula.

5.52 SearchForUnreferencedCells

Searches a specified area of a worksheet for cells aren't referenced by any formula.

This method is important because it can defend against some kinds of the most hard error to detect, errors of omission. A cell that contains a number that does not participate in any formula may have been left out of a model unintentionally.

APPLIES TO

IOAKAddIn, IOAKAPI

SYNTAX

result1 = expression1 .SearchForUnreferencedCells

Set result2 = expression2 .SearchForUnreferencedCells(Range1, formula, numeric, text, logical)

expression1 Required. An expression that returns an IOAKAddIn object.

expression2 Required. An expression that returns an IOAKAPI object.

result1 An OAKResult enumeration indicating the success or otherwise of the action.

result2 A Range containing of all the cells within Range1 that are not mentioned in at least one formula, or Nothing if no such cells are found.

Range1 Required Range. The range in which to search for cells that are not mentioned by any formula.

formula, numeric, text, logical Required Boolean. True if the method is to return cells of the various types listed.

REMARKS

When applied to an IOAKAddIn object, the SearchForUnreferencedCells method performs the same action as the OAK Development/Review | Search | Unreferenced cells command. It searches for cells within the current selection that are not mentioned by a formula in at least on other cell.

When applied to an IOAKAPI object, the SearchForUnreferencedCells method applies searches for cells within the the range indicated by the parameter submitted to the method that are not mentioned by a formula in at least on other cell.
It would be common to set the **formula** parameter to FALSE to prevent the method from selecting the formulas on the worksheet, some of which at least would be final outputs which by definition are not referenced by other formulas.

It would be common to set the **text** parameter to FALSE to prevent the method from selecting the text labels on the worksheet, which one would not expect to be referenced.

**WRINKLES**

When applied to an **IOAKAddIn** object, the **SearchForUnreferencedCells** method follows the One-cell rule. When applied to an **IOAKAPI** object, it does not: if supplied a Range parameter describing a single cell, it will look in that cell only.

When applied to an **IOAKAddIn** object, the method will display a dialog box before performing the search to ask what kinds of cells are to be selected, and another after the search if no unreferenced cells are detected, just as the equivalent OAK user-interface command does.

Both methods will report as unreferenced cells which, while not mentioned by any formula on the same worksheet, are mentioned by a formula on another worksheet.

### 5.53 **SearchForValue**

Searches a specified area of a worksheet for cells that fall into a range of values.

**APPLIES TO**

**IOAKAPI**

**SYNTAX**

```
Set result1 = expression1 .SearchForValue(Range1 , type , value0 , [ value1 ])
```

- **expression1** Required. An expression that returns an **IOAKAPI** object.
- **result1** A **Range** containing of all the cells within **Range1** that match **value0**, and may be **value1**; or **Nothing** if no such cells are found.
- **Range1** Required **Range**. The range in which to search for cells that are not mentioned by any formula.
- **type** Required **SearchMatchType** Indicates what kind of search to perform. One of **SearchMatchType_Contains**, **SearchMatchType_DoesNotContain**, **SearchMatchType_EndsWith**, **SearchMatchType_Equals**, **SearchMatchType_GreaterThan**, **SearchMatchType_GreaterThanOrEqualTo**, **SearchMatchType_IsBetween**, **SearchMatchType_IsNotBetween**,
SearchMatchType_LessThan, SearchMatchType_LessThanOrEqual, 
SearchMatchType_NotEquals, SearchMatchType_StartsWith.

value0  Required Variant. The value being searched for.

value1  Optional Variant. The value being searched for. Only needed when type is 
SearchMatchType_IsBetween.

REMARKS

The IOAKAddIn method that corresponds to the conditional search facilities offered by 
the OAK user interface is SearchFor. That method is offered for IOAKAPI too, but the 
combination of parameters only supports some of the search functionality. The 
functionality not conveniently implemented by SearchFor is caught by this 
SearchForValue method. For this reason, SearchFor is a method of both IOAKAddIn 
and IOAKAPI, but SearchForValue is a method of IOAKAPI only.

WRINKLES

When operated through the user interface, OAK's conditional search follows the One-Cell 
rule. The SearchForValue does not; if supplied a Range1 parameter describing a single 
cell, it will look in that cell only.

5.54 ShowProgress

Gets or sets a boolean value to specify whether user-interface progress indicators are 
shown during calls to IOAKAPI members.

APPLIES TO

IOAKAPI

SYNTAX

showProgress = expression .ShowProgress

expression .ShowProgress = True

expression  Required. An expression that returns an IOAKAPI object.

showProgress  A boolean value indicating the value of the property.

REMARKS

By default, user-interface progress indicators are not shown during calls to IOAKAPI 
members. The purpose of the IOAKAPI interface is to provide access to OAK functions
that bypass the user-interface. However, in many cases, the progress indicators may still be desired, for example, when a long-running operation has been initiated using parameters calculated in a VBA macro. In this case, the progress indicators can be turned on using this property.

5.55 Summarize

Generates a report giving essential statistics of a workbook or worksheet.

APPLIES TO

IOAKAddIn, IOAKAPI

SYNTAX

Set result1 = expression1.Summarize

Set result2 = expression2.Summarize(bookOrSheets, summarizeWorkbook, listDistinctFormulas, listReferredConstants, outputInSameWorkbook, [optional] riskAnalysis)

expression1 Required. An expression that returns an IOAKAddIn object.

expression2 Required. An expression that returns an IOAKAPI object.

result1 An OAKResult enumeration indicating the success or otherwise of the action.

result2 The Workbook in which the report has been placed.

bookOrSheets Required. One of
• an Excel workbook object
• an Excel worksheet object
• a collection of worksheet objects
• an array of worksheet objects

summarizeWorkbook Required Boolean. If True, causes OAK to include in its report a worksheet summarizing key statistics of the workbook or worksheets under investigation. If False, OAK does not generate that worksheet.

listDistinctFormulas Required Boolean. If True, causes OAK to include in its report a worksheet listing the unique formulas (ones that are not copies of their neighbours) in the workbook or worksheets under investigation. If False, OAK does not generate that worksheet.
**listReferredConstants** Required **Boolean**. If **True**, causes OAK to include in its report a worksheet listing the constants) in the workbook or worksheets under investigation. This list can then be checked against any model documentation. If **False**, OAK does not generate that worksheet.

**outputInSameWorkbook** Required **Boolean**. If **True**, causes OAK to append the worksheets that make up its report to the workbook on which it is reporting. If **False**, places those worksheets in a new workbook.

**riskAnalysis** Optional **Boolean**. If **True**, causes OAK to include in its report a worksheet listing the formulas found in the workbook, each with a count of a selection of formula elements it contains, which combined with a set of coefficients that are associated with those elements, produces an overall risk score for the workbook. If **False**, OAK does not generate that worksheet.

**REMARKS**

When applied to an **IOAKAddIn** object, the Summarize method activates the same dialog box as is presented when the Unhide cells command is selected manually from the OAK user interface. The action that follows is controlled by the data gathered by the dialog box from the user.

When applied to an **IOAKAPI** object, the Summarize method generates a report characterized by the parameters submitted to the method.

### 5.56 Transpose

Copies the formula(s) from the source range to the target range, without adjusting cell coordinates as Excel normally does, and transposing rows and columns

**APPLIES TO**

**IOAKAddIn**, **IOAKAPI**

**SYNTAX**

Set `result1 = expression1.Transpose(expression2).Transpose(source [, target])`

- **expression1** Required. An expression that returns an **IOAKAddIn** object.
- **expression2** Required. An expression that returns an **IOAKAPI** object.
- **result1** An OAKResult enumeration indicating the success or otherwise of the action.
- **source** Required **Range**. The cells to be copied.
**target**  Optional **Range**  The place where the cells are to be copied to. Normally just a single cell. If omitted, the is taken to be the top left corner of the source (which will result in overwriting of the some or all of the source).

**REMARKS**

When applied to an **IOAKAddIn** object, the Transpose method acts on the current selection in the same way as the Transpose command does. An exception may be raised if the selection is not suitable for the action.

When applied to an **IOAKAPI** object, the Transpose method acts on the ranges indicated by the parameters submitted to the method.

If the target overlaps the source, some or all of the source will be overwritten. That aside, the source is not altered by this action: the action is to copy cells, not cut them, and the originals are left in place.

**5.57  UndoCompareModifications**

Removes any rows and columns that have been inserted to align a worksheet by previous use of the OAK’s workbook/worksheet Compare facilities (either through the OAK user interface or by the VBA method).

**APPLIES TO**

**IOAKAddIn** , **IOAKAPI**

**SYNTAX**

```
result1 = expression1 .UndoCompareModifications
expression2 .UndoCompareModifications (bookOrSheet)
```

- **expression1**  Required. An expression that returns an **IOAKAddln** object.
- **expression2**  Required. An expression that returns an **IOAKAPI** object.
- **result1**  An OAKResult enumeration indicating the success or otherwise of the action.

**BookOrSheets**  Required. One of

- an Excel workbook object
- an Excel worksheet object
- a collection of worksheet objects
- an array of worksheet objects
REMARKS

When applied to an `IOAKAddIn` object, the `UndoCompareModifications` method activates the same dialog box as is presented when the Remove compare modifications is selected manually from the OAK user interface. The action that follows is controlled by the data gathered by the dialog box from the user.

When applied to an `IOAKAPI` object, the `UndoCompareModifications` method the workbook or worksheets to be changed are specified by the parameter submitted to the method.

5.58 UnhideCells

Exposes any hidden cells in the specified workbook or worksheets by making visible any hidden rows or columns.

APPLIES TO

`IOAKAddIn`, `IOAKAPI`

SYNTAX

```
Set result1 = expression1 .UnhideCells
expression2 .UnhideCells(BookOrSheets )
```

- `expression1` Required. An expression that returns an `IOAKAddIn` object.
- `expression2` Required. An expression that returns an `IOAKAPI` object.
- `result1` An OAKResult enumeration indicating the success or otherwise of the action.
- `BookOrSheets` Required. One of
  - an Excel workbook object
  - an Excel worksheet object
  - a collection of worksheet objects
  - an array of worksheet objects

REMARKS

When applied to an `IOAKAddIn` object, the UnhideCells method activates the same dialog box as is presented when the Unhide cells command is selected manually from the OAK user interface. The action that follows is controlled by the data gathered by the dialog box from the user.
When applied to an `IOAKAPI` object, the UnhideCells exposes any hidden cells on the workbook or worksheet or worksheets specified by the parameters submitted to the method.

EXAMPLE

To expose any hidden cells in the active worksheet:

```
MyOakApi.UnhideCells ActiveWorksheet
```

### 5.59 WorksheetManager

Activates the Worksheet Manager.

APPLIES TO

`IOAKAddIn`

SYNTAX

```
result1 = expression1 .WorksheetManager
```

- `expression1` Required. An expression that returns an `IOAKAddIn` object.
- `result1` An OAKResult enumeration indicating the success or otherwise of the action.

REMARKS

When applied to an `IOAKAddIn` object, the WorksheetManager method activates the same dialog box as is presented when the Worksheet Manager command is selected manually from the OAK user interface. The action that follows is controlled by the data gathered by the dialog box from the user, and acts on the workbook or worksheet(s) specified in that dialog.
Known limitations
6.1 Known limitations

OAK can't yet handle

- multisheet (3D) references
- structured references.
- doubly prefixed references

In addition, there are some limits to the diversity of cell formatting that Excel can accommodate, that OAK may hit in certain circumstances.

6.2 3-D references

Excel allows formulas to aggregate information from a range of worksheets, as in

The parser built in to OAK can now make sense of this kind of reference, but the logic that processes sum expansion mishandles them. If OAK encounters one, it will report that "An error has occurred in OAK", or generate a result that is clearly incorrect.

This issue can affect any of the functions that use OAK's Excel formula language parser, specifically

- OAK Review | Formula | Reconstruct
- OAK Development | Formula | Optimize
- OAK Review | Formula | Prune
- OAK Development | Names | Redefine
- OAK Development | Names | Recreate
- OAK Development | Names | Deapply

Operis is working on fixing this as these formulas are reasonably common, and because they are hard to decipher: an ability of reconstruction and discrepancy analysis to study these expressions would be particularly valuable.

6.3 Structured references

Excel 2003 introduced, and Excel 2007 has developed, a notion of structured references, a means of identifying the elements of a table that are to be acted on by a formula.

The parser built in to OAK does not handle this kind of reference. If it encounters one, it will report that "An error has occurred in OAK".

This issue can affect any of the functions that use OAK's Excel formula language parser, specifically

- OAK Review | Formula | Reconstruct
Operis intends to fix this issue in due course, but does not consider it urgent as it has very rarely encountered a spreadsheet prepared by a client that uses this feature.

If you disagree with this perception and find this limitation troublesome, please let Operis know by email to oak@operis.com.

### 6.4 Doubly prefixed references

The expression `=SUM(Sheet1!A1:B5)` can be written as `=SUM(Sheet1!A1:Sheet1!B5)`. The redundancy adds nothing, but the expression is legal in Excel.

The parser built in to OAK does not handle this kind of reference. If it encounters one, it will report that "An error has occurred in OAK".

This issue can affect any of the functions that use OAK's Excel formula language parser, specifically:

- OAK Review | Formula | Reconstruct
- OAK Development | Formula | Optimize
- OAK Development | Formula | Prune
- OAK Development | Names | Redefine
- OAK Development | Names | Recreate
- OAK Development | Names | Deapply

Operis is working on fixing this issue. Until it does, the workaround is to alter such expressions to remove the second prefix in the reference.

### 6.5 Format limits

OAK's worksheet comparison and workbook mapping commands have the option to display what they have found by applying color to the worksheet(s) being examined. Remove Color Formatting and Remove Compare Modifications also manipulate fonts and colors.

Excel cannot handle a workbook in which every cell is a different color, typeface and font size, and also has limits on the number of hyperlinks that can appear in a sheet. Excel
Reference material: Known limitations

keeps a list of all the different combinations of cell formats in use in a workbook, and beyond some limit will allow no further permutations of formatting to be added because the list has a fixed size and has become full.

OAK may take Excel beyond this limit when it attempts to mark worksheets that are already busy in formatting terms, since OAK achieves the marking by applying appropriate formatting to cells. In such cases OAK will stop the operation prematurely, and issue a warning that it has done so.

Operations that seemingly reduce the format count, such as Remove Color Formatting and Remove Compare Modifications can encounter this error, because they work by applying a format that is considered non-colorful, sheet by sheet, and this can generate a temporary increase in the format count.

This limitation is a lot worse prior to Excel 2007, and details of it can be found at http://support.microsoft.com/kb/213904.

In the meantime, workarounds can be found under the subheading “Wrinkles” for the commands affected.

6.6 Deleting corrupt names

Sometimes OAK cannot delete/recreate names in Excel.

Sometimes it is possible to delete these names manually using Formulas | Defined Names | Name Manager. Other times it may be possible by switching to R1C1 style (Office Button | Excel Options | Formulas | Working with formulas), at which time Excel will prompt the user to change each corrupt name.

See http://support.microsoft.com/kb/555127.
Problem solving
7.1 Problem solving

Operis has listed in this help

- how to instruct the OAK setup program to display the results of the tests it performs to see if OAK is likely to work in the environment provided by a workstation
- common problems, in the form of OAK behaviours that are slightly counterintuitive and can appear to be bugs but are not.

If you encounter issues that are not covered in either of these categories, Operis would like to hear from you.

7.2 Forcing preinstallation testing

The largest users of spreadsheets are found in the financial services sector. Security is generally tight in businesses in this line of work, since that is where the money is; and this extends to the workstations used there, which are frequently locked down in ways that intentionally present obstacles to the installation of new software.

Operis has therefore paid particular attention to making the utility responsible for installing the product as robust as possible to awkward situations. The setup program has been engineered to perform a variety of preinstallation tests, to inspect the environment in which it finds itself and form an impression (but not a certainty) of the likelihood that OAK will run without problems. These tests are described in the part of this help that describes installing OAK.

By default, the installer displays the results of the preinstallation tests only if one or more of the tests is failed. It can however, be instructed to display the results of the test whether or not they are passed.

To activate this preinstallation test mode, use following command substituting the actual name of the installer file for OAK4.msi:

```bash
msiexec /i OAK4.msi PRETEST=true
```

This can be done

- either, by typing that command at the Windows Command Line, after making current the directory in which the MSI file has been placed
- or, easier, put that line in a .bat file, place it in the same directory as the msi file, and double click on it to run it.

7.3 Common problems

MAP WORKBOOK GIVES INCOMPLETE RESULT

This is a limitation of Excel, not OAK. It is discussed further under known limitations.
COMPARE GIVES INCOMPLETE RESULT

This is a limitation of Excel, not OAK. It is discussed further under known limitations.

FORMULA | RECONSTRUCT DELIVERS FEWER LEVELS OF PRECEDENT SUBSTITUTION THAN SPECIFIED

This behavior is by design and is discussed in the article called Mixed levels within OAK Concepts.

THE CONFINE TO ACTIVE PATH OPTION OPTION ON THE FORMULA | RECONSTRUCT DIALOG IS GRAYED OUT

This option is only offered if the selection consists of a single cell.

FORMULA | PRUNE MAKES NO CHANGE TO A FORMULA

OAK Review | Formula | Prune will turn the formula =IF(A1>B2,C3,D4) into either =C3 or =D4, depending on whether A1 is bigger than B2 or not. In order to know which option to take, OAK has to evaluate the logical expression A1>B2. If that results in an error, OAK will be unable to make a choice.

The error is most likely to be the legitimate consequence of the values that happen to be in the spreadsheet; A1 or B2 could be #DIV/0, it could be due to limitations of Excel when asked to evaluate the expression, or a defect in OAK. Whatever the cause, OAK will do nothing, leaving the formula unchanged.

7.4 Reporting bugs

Operis would like to hear about it if OAK does not perform as expected, or if the documentation is inaccurate. Please direct bug reports to oak@operis.com. But first, do look at the Common problems page.

If your problem is that OAK won’t even install, please include a report generated from the preinstallation test dialog.

If the issue you wish to report involves OAK crashing, details of what has gone wrong are stored in the Windows Event Viewer. Operis will find it valuable to receive this material.

1 Look in the Windows event viewer (Control Panel | Administrative tools | Event viewer)
2 Find the OAK error (View | Filter may help)
3 Double click on it
4 Copy the event description to the clipboard (bottom of three buttons)
5 Paste the result in your report email.
7.5 Finding where OAK is installed

Some actions, notably installing the latest help files, required knowledge of which folder OAK is stored in.

OAK is usually located in the directory C:\Program Files\Operis\OAK4, but it might be somewhere else if the defaults suggested were overridden when the product was installed.

The simplest way is to look in the COM add-ins dialog. When OAK is selected, the location is displayed in the dialog.

An alternative is to search your hard drive for the file Operis.OAK.dll and noting which folder it turns up in.

If the file cannot be found on the hard drive, OAK may have been installed on a network drive. You could try searching there.

If you still can't find the OAK4 folder, or the network drives are too large or numerous to search in any reasonable time, another way to find it is to

- invoke the help that comes with OAK (OAK Development/Review | Help)
- drop the control in the top left of the window (click on the icon at the left edge of the title bar, or press Alt+Space)
- choose the Jump to URL... option
- read the address of the folder from the dialog that results.
Tip: The URL will probably be only partly visible, but the concealed part can be exposed by selecting the upper field of the dialog and navigating with the left and right keys or the mouse.

For example, the Current URL mk:@MSITStore:C:\Program%20Files\Operis\OAK4\OAK4.chm::/chapterwelcometoak.htm shows that OAK has been installed in C:\Program Files\Operis\OAK4.

Tip: The space in "\Program files:" is shown as %20 because the URL is displayed using Percent-encoding. There is a Wikipedia article that describes this coding, but the path can probably be understood without deciphering it exactly.

7.6 Disabled Items

Sometimes Excel will associate an add-in with an error that has occurred, and take the precaution of disabling it. This disabling does not simply deactivate it, because if a user tries to reactivate it via one of the add-in managers, it will still not work.

To determine if an add-in has been placed in the disabled items list, go to the Add-ins Manager:

Notice the item in the "Disabled Application Add-ins" section of the list. To make this add-in available for use again, select "Disabled Items" in the drop down list, and click the Go button.

Excel will then show the "Disabled Items" dialog. Select the add-in you want to enable, and click the "Enable" button.
7.7 Licensing

When most users of OAK purchase a license online, they will be sent a serial number to enter into the licensing form, and activate online, according to the instructions in the section: Buying OAK

There are other situations that can arise. There may be a problem storing the license, or if a site license has been purchased a site license file will have been provided.

This may seem rather complicated, as OAK 4.3 and OAK 4.4 are in a transition between an older licensing system and a newer system that is part of an online purchasing and activation process. Single user licenses are now stored at the user level instead of the machine level, although licenses are still considered machine level.

COM REGISTRATION

OAK uses a 3rd-party licensing component that requires a COM registration, which is done by the installer. In some IT environments with heightened security or system maintenance, the COM registrations performed during the installation process get deleted. IT staff in such organizations need to make sure these registrations do not get deleted after OAK is installed.

OAK LICENSE STORAGE

When installed, OAK 4.0 to 4.2 create a folder Operis\OAK4 in the Common Application Data folder. The folder that OAK creates will be referred to as the OAK Common Data folder. License files (referred to as legacy license files) for OAK 4.1 and 4.2 are stored in this folder, and can remain once a user has upgraded to OAK 4.3, in which they are still supported.

The default location of the OAK Common Data folder in Windows Vista and later is C:\ProgramData\Operis\OAK4.

Files and folders stored in this location requires, by default, administrator rights to write to them. In OAK 4.1 and 4.2, it was necessary for an administrative user to install the license key.
With OAK 4.3, this was deprecated in favor of using local and roaming user application data folders, with the assumption that the vast majority of users are the only user of the machine on which they run OAK. These folders are specified in the table below, using the appdata environment variable to specify the actual location on the system. To go there, simple enter the %appdata% along with the rest of the location in the Windows Explorer address bar and it will be expanded to whatever value it is for the current user.

<table>
<thead>
<tr>
<th>OAK Application Data Folder</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Roaming User</td>
<td>%appdata%\Operis\OAK4</td>
</tr>
<tr>
<td>Local User</td>
<td>%appdata%..\Local\Operis\OAK4</td>
</tr>
</tbody>
</table>

As of version 4.3, when OAK starts up, it searches the OAK Local User Application Data folder for a legacy licenses (license.lic) or a single user license (OAK4license.lic).

If it doesn't find a legacy license there, OAK will look in the OAK Common Application Data folder for a legacy license file, and if it finds one, it will copy it to the OAK Local User Application Data Folder.

If it doesn't find a single user license there, OAK will copy the license file it is deployed with to the OAK Local User Application Data folder.

OAK will also search for site license. It will search the OAK Roaming User Application Data folder and the OAK Common Application Data folder for all the files with the .slic extension it can find, order them based on creation date and time, then search them newest to oldest for one that is current.

**LICENSING THROUGH THE OAK USER INTERFACE**

As of OAK 4.3, modification of license files, both single user license files and the legacy license files are done in the user's local application data folder, so by default, no administrator rights are required.

**MULTIPLE USERS, SAME MACHINE**

OAK4 is still licensed on a per-machine basis, but as of OAK 4.3, the workflow is oriented toward the single-user per machine scenario. You are recommended to contact Operis if this is a problem.

**SITE LICENSES**

If OAK is licensed with a site license, then a binary file with a .slic extension will have been provided. This contains encrypted information about the company who has the license, a branding logo, and the expiry date.

This file should be copied into OAK's common data folder.
Many organizations use a software deployment system, and a site license file could be delivered to the required location as part of a deployment package.

However, as stated in a previous paragraph, site license files can also be stored in the user's roaming application data folder.

### 7.8 COM Registrations

OAK is a COM add-in, and the 3rd party licensing component it uses, is a COM component. These require COM registrations, which the OAK installer does when it is used to install OAK.

Some users of OAK have found the following:

- OAK works immediately after installation

After a restart, or regular system maintenance, OAK does at least one of:

- Stops appearing in Excel

- Displays an error message saying "OAK was not able to start a 3rd-party component it uses for licensing"

If you encounter either of these problems, you need to ask your organization's IT staff to make sure that the COM registrations that are set up by OAK's installer package do not get removed by security or system maintenance software, or by them when they repackage OAK for your IT environment. See the section titled OAK Registry Settings for details of some of these COM registrations.
7.9 OAK Registry Settings

This section describes how Excel is configured to load OAK, and how the installer makes this happen. It is intended for IT administrators and power users.

When OAK is installed, the installer performs COM registrations at machine level and configures Excel to install OAK in the COM add-ins list of each user as they log in and start Excel. It also installs a 3rd party licensing component and causes it to perform COM self-registration.

When OAK is uninstalled, the installer removes the COM registrations, configures Excel to remove the registry keys that install OAK in the COM add-ins list of each user as they log in and start Excel, unregisters and removes the 3rd party licensing component.

OAK COM REGISTRATION

OAK has various COM registrations:

- The OAK Add-in COM registrations
- The licensing component COM registrations
- The OAK COM API type library registrations

This section covers the first two. Throughout this section, we use Windows regedit scripts to show registry entries that are added or deleted.

THE OAK ADD-IN COM REGISTRATION

The GUID default value of the Operis.OAK.Connect\CLSID key refers to an entry in the Classes\CLSID key:

32-bit Windows, 32-bit COM registration

```
Windows Registry Editor Version 5.00

[HKEY_LOCAL_MACHINE\SOFTWARE\Classes\Operis.OAK.Connect\CLSID]
@="{355D8B61-817A-48D1-81AC-FFDA39CA5F77}"

[HKEY_LOCAL_MACHINE\SOFTWARE\Classes\CLSID\{355D8B61-817A-48D1-81AC-FFDA39CA5F77}\]
@="Operis.OAK.Connect"

[HKEY_LOCAL_MACHINE\SOFTWARE\Classes\CLSID\{355D8B61-817A-48D1-81AC-FFDA39CA5F77}\InprocServer32]
@="C:\\Program Files\\Operis\\OAK4\\\Operis.OAK.AddIn.dll"
"ThreadingModel"="Apartment"
```
64-bit Windows has both a 32-bit and 64-bit COM registration, routing the instantiation to different shim DLLs depending on the bitness.

64-bit Windows, 32-bit COM registration

Windows Registry Editor Version 5.00

[HKEY_LOCAL_MACHINE\SOFTWARE\Classes\CLSID\{355D8B61-817A-48D1-81AC-FFDA39CA5F77}\ProgID]
@="Operis.OAK.Connect"

[HKEY_LOCAL_MACHINE\SOFTWARE\Classes\CLSID\{355D8B61-817A-48D1-81AC-FFDA39CA5F77}\Programmable]

64-bit Windows, 64-bit COM registration

Windows Registry Editor Version 5.00

[HKEY_LOCAL_MACHINE\SOFTWARE\Classes\WOW6432Node\Operis.OAK.Connect\CLSID]
@="{355D8B61-817A-48D1-81AC-FFDA39CA5F77}"

[HKEY_LOCAL_MACHINE\SOFTWARE\WOW6432Node\Classes\CLSID\{355D8B61-817A-48D1-81AC-FFDA39CA5F77}\InprocServer32]
@="C:\\Program Files (x86)\\Operis\\OAK4\\Operis.OAK.AddIn.dll"
"ThreadingModel"="Apartment"

[HKEY_LOCAL_MACHINE\SOFTWARE\WOW6432Node\Classes\CLSID\{355D8B61-817A-48D1-81AC-FFDA39CA5F77}\ProgID]
@="Operis.OAK.Connect"

[HKEY_LOCAL_MACHINE\SOFTWARE\WOW6432Node\Classes\CLSID\{355D8B61-817A-48D1-81AC-FFDA39CA5F77}\Programmable]
EXCEL USER SETTINGS, ON INSTALLATION

Windows is a multi-user operating system, yet OAK installs at the machine level, and the OAK COM class can be instantiated by any user. For Excel to load OAK, further configuration is required. This can be done at the machine level, but then the users would need to acquire administrator privileges to activate or deactivate OAK, and if they were to do this, OAK would be activated or deactivated for all users. So OAK needs to be installed in Excel on a per-user basis. However, the current user's registry settings are not available at install time, nor are the Excel registry settings for all the other users.

Office 2007 and later provide a set of registry keys called User Settings to instruct Office applications including Excel on registry keys to create or delete when each user logs in and starts the application.

The OAK installer .msi uses these settings, on installation, to specify that Excel should set up certain user-level registry keys when a user starts it, and on uninstallation of the .msi, it replaces these keys with instructions to delete these user-level registry keys.

The location of these settings are different depending on the version (2007/12.0, 2010/14.0, 2013/15.0, 2016/16.0), the edition (32-bit, 64-bit) and the edition of Windows (32-bit, 64-bit). Also, the content can vary.

Key items to notice here are:

- **version** - This is the version of Office. 12.0 for Office 2007, 14.0 for Office 2010, 15.0 for Office 2013, 16.0 for Office 2016

- **count** - This is effectively a version number for the Operis_OAK4 key, and used by Excel to determine a user's local registry is up-to-date.

HKLM\SOFTWARE\Wow6432Node or HKLM\SOFTWARE - The presence of Wow6432Node in the location indicates keys for 32 bit software in 64-bit Windows.

32-bit Windows with 32-bit Excel and 64-bit Windows with 64-bit Excel

Windows Registry Editor Version 5.00

[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Office\version\User Settings\Operis_OAK4]
"Order"=dword:00000001
"Count"=dword:count

[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Office\version\User Settings\Operis_OAK4\Create]

[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Office\version\User Settings\Operis_OAK4\Create\SOFTWARE\Microsoft\Office\Excel\Addins\Operis.OAK.Connect]
"Description"="The Key to Spreadsheet Correctness"
"FriendlyName"="Operis Analysis Kit 4.4"
"LoadBehavior"=dword:00000003
Reference material: Problem solving

64-bit Windows, 32-bit Excel

Windows Registry Editor Version 5.00

[HKLM\SOFTWARE\WOW6432Node\Microsoft\Office\version\User Settings\Operis_OAK4]
"Order"=dword:00000001
"Count"=dword:count

[HKLM\SOFTWARE\WOW6432Node\Microsoft\Office\version\User Settings\Operis_OAK4\Create\SOFTWARE\Microsoft\Office\Excel\Addins\Operis.OAK.Connect]
"Description"="The Key to Spreadsheet Correctness"
"FriendlyName"="Operis Analysis Kit 4.4"
"LoadBehavior"=dword:00000003

The OAK installer installs user settings for all versions of Excel that it finds.

USING OAK

Following installation by .msi, when each user of the machine starts Excel, Excel will look at HKCU\Software\Wow6432Node\Microsoft\Office\version\User Settings\Operis_OAK4 for the "Count" value. If this is less than the "Count" value in the corresponding HKLM\SOFTWARE\WOW6432Node\Microsoft\Office\version\User Settings\Operis_OAK4, Excel will update the user's local registry accordingly.

The "Create" keys cause the following registry keys and values to be created:

32 bit Windows with 32-bit Excel and 64-bit Windows with 64-bit Excel

HKCU\SOFTWARE\Microsoft\Office\Excel\Addins\Operis.OAK.Connect]
"Description"="The Key to Spreadsheet Correctness"
"FriendlyName"="Operis Analysis Kit 4.4"
"LoadBehavior"=dword:00000003

64-bit Windows with 32-bit Excel

HKCU\SOFTWARE\Wow6432Node\Microsoft\Office\Excel\Addins\Operis.OAK.Connect]
"Description"="The Key to Spreadsheet Correctness"
"FriendlyName"="Operis Analysis Kit 4.4"
"LoadBehavior"=dword:00000003

- The key "Operis.OAK.Connect" is the ProgID of the OAK COM class. Excel uses this to create the OAK COM object (the add-in).

- The description and friendly name are used in the add-ins managers.

- The LoadBehavior value must be 3 for Excel to load OAK on startup. If a user disables OAK in their COM add-ins manager, this value will be changed.
EXCEL USER SETTINGS, ON UNINSTALLATION

On uninstallation, the OAK installer removes the "Create" key, and replaces it with a "Delete" key that specifies that Excel should remove those items from each user's local registry. The count value is incremented to show Excel that there's an update to the user settings.

32-bit Windows with 32-bit Excel and 64-bit Windows with 64-bit Excel

```
Windows Registry Editor Version 5.00

[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Office\version\User Settings\Operis_OAK4]
"Order"=dword:00000001
"Count"=dword:count+1

[-HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Office\version\User Settings\Operis_OAK4\Create]

[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Office\version\User Settings\Operis_OAK4\Delete\SOFTWARE\Microsoft\Office\Excel\Addins\Operis.OAK.Connect]
```

64-bit Windows, 32-bit Excel

```
Windows Registry Editor Version 5.00

[HKEY_LOCAL_MACHINE\SOFTWARE\WOW6432Node\Microsoft\Office\version\User Settings\Operis_OAK4]
"Order"=dword:00000001
"Count"=dword:count+1

[-HKEY_LOCAL_MACHINE\SOFTWARE\WOW6432Node\Microsoft\Office\version\User Settings\Operis_OAK4\Create]

[HKEY_LOCAL_MACHINE\SOFTWARE\WOW6432Node\Microsoft\Office\version\User Settings\Operis_OAK4\Delete\SOFTWARE\Microsoft\Office\Excel\Addins\Operis.OAK.Connect]
```

LICENSING COMPONENT COM REGISTRATIONS

On installation, OAK copies a file called LicProtector NNN .EXE (where NNN is a version number such as 500) to %WINDIR%\System32 in 32-bit Windows, or %WINDIR%\SysWow64 on 64-bit Windows. This executable is a 32-bit COM server. Once OAK has copied it to the target location, it executes it as follows to have it register itself.

```
LicProtector NNN.exe /regserver /NOREDIRECT
```

On uninstallation, OAK executes the file again:

```
LicProtector NNN.exe /unregserver /NOREDIRECT
```
CONCLUSION

This section covered the COM registration of OAK as a COM component, setting up of Office User Settings to add and remove OAK from each user's COM add-ins list, and a briefly, how the licensing component is registered and unregistered during the installation and uninstallation process.

This section does not cover the registry settings that are created to register the OAK type library for programming with OAK via its COM API, for example, in VBA.

Hopefully this section is helpful to IT staff when security settings in an environment prevent OAK from being installed properly. You should be warned that although it can appear you have solved a problem by adding OAK to Excel manually, or running LicProtector NNN .exe /regserver manually to register it for the current user, this creates a configuration that is out of sync with the installer, and so will not be undone when OAK is uninstalled.
Resources
8.1 Useful spreadsheet resources

Among resources that OAK users might find valuable are

- Eusprig
- Ray Panko's spreadsheet research website
- Microsoft's Excel blog

8.2 Eusprig

Eusprig is the European Spreadsheet Risks Interest Group. It runs a well-established annual conference which provides a forum for researchers, practitioners, trainers, vendors, consultants and auditors.

Now having run for over 17 years, the conference has resulted in the strongest collection of peer-reviewed papers on spreadsheet risks and how to mitigate them.

Eusprig also maintains an email list at eusprig.yahoo.groups.

SEE ALSO

Eusprig's website is at http://www.eusprig.org/

8.3 Ray Panko's spreadsheet research website

Ray Panko is the world authority on spreadsheet errors. A professor at the University of Hawaii, he has been researching the subject since the 1960s.

His research is distinctive in that it is driven by measurement. On the question, for example, of whether it is good to express formulas in terms of meaningful names or the more traditional coordinate notation, most practitioners of spreadsheet modelling would have a strongly held view. Ray Panko would not. He would gather several dozen students, make half try one way and half try the other, and see whether there was a difference in error rates of any statistical significance.

His key finding is that the error rates in spreadsheets, in software development and other cognitive endeavours are all of the same nature and order, which leads him to the conclusion that practices already proved in the development of software for safety-critical systems can be expected to be useful in the delivery of robust spreadsheets.

SEE ALSO

Ray Panko's spreadsheet research website is at http://www.panko.com/ssr/index.html

8.4 Microsoft Excel blog

The Microsoft Office development team operates a blog. It has interesting articles about
Excel's current and planned capabilities.

SEE ALSO

The Microsoft Excel blog is at https://blogs.office.com/excel/
Version histories
## 9.1 OAK version history

<table>
<thead>
<tr>
<th>Version</th>
<th>Released</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.40.035</td>
<td>24 March 2019</td>
<td>Public release based on 4.40.033</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Fix for failure of summarize tool to report a parse error instead of crashing.</td>
</tr>
<tr>
<td>4.40.034</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>4.40.033</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Restricted public release based on 4.40.030, made available to some users</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Added diagnostic information for a formula parsing error</td>
</tr>
<tr>
<td>4.40.032</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>4.40.031</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>4.40.030</td>
<td>19 September 2018</td>
<td>Public release based on 4.40.024</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Fixed a problem with quotation marks in sheet names in map operation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Fixed an error that occurred when pruning multiple ranges.</td>
</tr>
<tr>
<td>4.40.028-029</td>
<td>Versions internal to Operis in which its consultants trial the changes in later external releases.</td>
<td></td>
</tr>
<tr>
<td>4.40.027</td>
<td>&quot;Rough cut&quot; release of future features.</td>
<td></td>
</tr>
<tr>
<td>4.40.026</td>
<td>Version internal to Operis in which its consultants trial the changes in later external releases.</td>
<td></td>
</tr>
<tr>
<td>4.40.025</td>
<td>&quot;Rough cut&quot; release of future features.</td>
<td></td>
</tr>
<tr>
<td>4.40.024</td>
<td>26 June 2018</td>
<td>Public release based on 4.40.018</td>
</tr>
</tbody>
</table>
- Fixed a failure to count references in the reference tracker

4.40.020-23

Versions internal to Operis in which its consultants trial the changes in later external releases.

4.40.019

"Rough cut" release of future features

4.40.018 1 January 2018

Public release based on 4.40.017

- Removed result-limiting optimisation in reconstructor

4.40.017 16 November 2017

Public release based on 4.40.015

- Fixed problem with non-reference names crashing the cross reference report
- Changed to FIPS-compliant decryption throughout

4.40.016

Reserved

4.40.015 23 October 2017

Public release based on 4.40.013

- Changed to FIPS-compliant decryption for reading site licenses.

4.40.014

Reserved

4.40.013 2 October 2017

Public release based on 4.40.007

- Fixed title of Fan Out Precedents options form.
- Improved error reporting and preinstallation checking in multiplatform installer.

4.40.011-12

Reserved.

4.40.010 3 September 2017

"Rough cut" release of future features.
4.40.008-009  Versions internal to Operis in which its consultants trial the changes in later external releases.

4.40.007 10 July 2017  Public release based on 4.40.003
- Improved error handling in summary risk analysis
- Restored COM/VBA API for the R1C1 vs A1 range comparison.
- Adjusted installer setup for 32-bit Excel.

4.40.004-006  Versions internal to Operis in which its consultants trial the changes in later external releases.

4.40.003 6 June 2017  Public release based on 4.40.001
- Merged 32 and 64 bit editions into a single msi file.
- Fixed wildcard search and replace in worksheet manager.
- Fix for Desktop Edition being restricted to Essentials limits.

4.40.002  Version internal to Operis in which its consultants trial the changes in later external releases.

4.40.001 13 May 2017  Public release based on 4.40.000
- Fixed issue with activation of 4.3 serial number not being persisted.

4.40.000 27 April 2017  Initial public release of OAK 4.4 series
- Two licensing levels - Essentials / Professional
- Enhanced color options in Map tool
- New workbook reference report in summary tool
- Updated to use the .NET 4.5.2 Framework
- Removed dependence on MS-Office Primary Interop Assemblies

4.30.038-042 Versions internal to Operis in which its consultants trial the changes in later external releases.

4.30.037 "Rough cut" release of OAK 4.4 features.

4.30.035, 036 Versions internal to Operis in which its consultants trial the changes in later external releases.

4.30.034 14 January 2017 Public release based on 4.30.030
- Added R1C1 comparisons to range compare tool
- Fixed error comparing ranges
- Summarize more robust to errors in risk analysis feature
- Updated CompareRanges API functions to handle R1C1 formulas.

4.30.032, 033 Versions internal to Operis in which its consultants trial the changes in later external releases.

4.30.031 "Rough cut" release of OAK 4.4 features.

4.30.030 3 May 2016 Public release based on 4.30.013
- Updated to use Microsoft .NET 3.5 Framework
- Improvements in license activation process
- Additional pre-installation check
- SHA256 code signing

4.30.015, 017,019,0 Versions internal to Operis in which its consultants trial the changes in later external releases.
"Rough cut" releases of OAK 4.4 features

4.30.014, 016,018,0 20,022,02 4,026,028

4.30.013 22 January 2016 Public release based on 4.30.012
- Optimization in interaction with Excel

4.30.012 Hot-fix release based on 4.30.009
- Fixed problem using COM/VBA API in non-English Excel

4.30.011 Version internal to Operis in which its consultants trial the changes in later external releases.

4.30.010 "Rough cut" release of OAK 4.4 features.

4.30.009 9 November 2015 Public release based on 4.30.006
- Simplified license activation for email and phone activations
- Updated advertising for Operis financial modelling book
- Removed feedback form as it wasn't being used.
- Bug fixes:
  - Data read from worksheet offset when cell A1 is merged
  - Bytes downloaded label on update progress form too short.

4.30.008 Version internal to Operis in which its consultants trial the changes in later external releases.
<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.30.007</td>
<td>&quot;Rough cut&quot; releases of OAK 4.4 features.</td>
<td></td>
</tr>
<tr>
<td>4.30.006</td>
<td>30 July 2015</td>
<td>&quot;Rough cut&quot; releases of OAK 4.4 features.</td>
</tr>
<tr>
<td></td>
<td>Bug fixes:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Slightly more informative error reporting regarding license errors</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Map colors not working for names and print area</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- VBAProjects remaining after workbooks processed by OAK are closed</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Error creating report with more than 255 worksheets.</td>
<td></td>
</tr>
<tr>
<td>4.30.001,</td>
<td>&quot;Rough cut&quot; releases of OAK 4.4 features.</td>
<td></td>
</tr>
<tr>
<td>003,005</td>
<td>Versions internal to Operis in which its consultants trial the changes in</td>
<td></td>
</tr>
<tr>
<td></td>
<td>later external releases.</td>
<td></td>
</tr>
<tr>
<td>4.30.000</td>
<td>17 February 2015</td>
<td>Initial public release of OAK 4.3 series.</td>
</tr>
<tr>
<td></td>
<td>Summarize: new risk analysis report</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Compare: name comparison, advanced alignment, reports type of change and</td>
<td></td>
</tr>
<tr>
<td></td>
<td>alignment details, detects changes in array dimensions.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Prune inactive path prunes more functions</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Deapply names now has option to ignore named constants</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Build Names Database shows name visibility</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Increased robustness and improved cancellability of long running map,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>summarize and compare operations.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>New Desktop Edition that runs OAK outside of Excel for improved memory use.</td>
<td></td>
</tr>
</tbody>
</table>
• Changed references to "hardwired constants" to "hardcoded constants" in API.

4.20.062-067 Versions internal to Operis in which its consultants trial the changes in later external releases.

4.20.061 11 September 2014 "Rough cut" releases of OAK 4.3 features.

4.20.059-060 Versions internal to Operis in which its consultants trial the changes in later external releases.

4.20.058 28 August 2014 "Rough cut" releases of OAK 4.3 features.

4.20.053-057 Versions internal to Operis in which its consultants trial the changes in later external releases.

4.20.052 11 February 2014 "Rough cut" release of OAK 4.3 features.

4.20.051 22 November 2013 Public release based on 4.20.050
• Fix for type library registration.
• Fix for error when clicking some column headers in the worksheet manager.

4.20.050 14 October 2013 Public release based on 4.20.038
• Fix for failure of summarize tool to list distinct formulas when that is all that is requested of it.
• Updated Operis address details in various places.
• Added references to Excel 2013 and Windows 8 to the documentation.

4.20.039, 041-043,045,047,049

Versions internal to Operis in which its consultants trial the changes in later external releases.

4.20.040, 044,046,048

"Rough cut" releases of OAK 4.3 features.

4.20.038 22 December 2012

Public release based on 4.20.022

• Enabled user-local storage of site license files.

• Updated installer for Windows 8

• Changed UI and documentation references to "hardwired constants" to "hardcoded constants"

4.20.037

• Modified comparison to detect more detail in date and currency formatted cells.

4.20.023-025,027-029,031-033,035

Versions internal to Operis in which its consultants trial the changes in later external releases.

4.20.023, 026,030,034,036

"Rough cut" releases of OAK 4.3 features.

4.20.022 26 June 2012

Public release of OAK mostly based on 4.20.009

• Significantly reduced memory use in the summarize tool

• Fixed:

  o Second summary with the report in the same workbook fails
- Summarize reports all prefixed cell references as off-sheet, even if they are not.
- OAK operations performed with Excel's events enabled.
- Worksheet Manager selection tool fails to recognize an open-ended range (e.g. 1-)

4.20.011-016, 4.20.018-020

Versions internal to Operis in which its consultants trial the changes in later external releases.

4.20.010, 017,021

"Rough cut" releases of OAK 4.3 features.

4.20.009 8 Feb 2012 Public release based on 4.20.008
- Fix for a problem with the Worksheet Manager which leaves calculation switched off.

4.20.008 8 Feb 2012 A version based on 4.20.000 for customers who hold site licenses for OAK.
- New Operis colors and logo

4.20.002, 003,005

Versions internal to Operis in which its consultants trial the changes in later external releases.

4.20.001, 004,006

"Rough cut" releases of OAK 4.3 features.

4.20.000 10 Oct 2011 Initial public release of OAK 4.2 series.
- Public release of Worksheet Manager

4.10.022-023 21 Sept 2011 "Rough cut" releases of OAK 4.2 features:
- Various improvements to the compare ranges tool and reports.
4.10.021  Version internal to Operis in which its consultants trial the changes in later external releases.

4.10.020  29 July 2011  "Rough cut" releases of OAK 4.2 features:
• Minor changes to compare tool.

4.10.019  Version internal to Operis in which its consultants trial the changes in later external releases.

4.10.018  19 June 2011  "Rough cut" release of OAK 4.2 features:
• Fixed problem pruning IF with references to blank cells.
• Documentation largely complete.

4.10.016-017  Version internal to Operis in which its consultants trial the changes in later external releases.

4.10.015  29 May 2011  "Rough cut" release of OAK 4.2 features:
• Added awareness of Excel bits to installer and update mechanism.

4.10.014  19 May 2011  "Rough cut" release of OAK 4.2 features.

4.10.012  1 April 2011  A version based on 4.00.010 for companies who hold site licenses for OAK.

4.10.011  21 April 2011  "Rough cut" release of OAK 4.2 features.

4.10.003-010  Versions internal to Operis in which its consultants trial the changes in later external releases.

4.10.002  20 July 2010  "Rough cut" release of OAK 4.2 features:
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4.10.001</td>
<td>11 June 2010</td>
<td>&quot;Rough cut&quot; release of 4.2 features.</td>
</tr>
<tr>
<td>4.10.000</td>
<td>12 June 2010</td>
<td>Initial public release of OAK 4.1 series.</td>
</tr>
<tr>
<td>4.00.024</td>
<td></td>
<td>A version internal to Operis in which its consultants trial the changes in later external releases.</td>
</tr>
<tr>
<td>4.00.023</td>
<td>28 May 2010</td>
<td>&quot;Rough cut&quot; release of OAK 4.1 and 4.2 features:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Added OAK 4.1 documentation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Added workaround for Excel's out of memory error on map/summarize/compare</td>
</tr>
<tr>
<td>4.00.022</td>
<td></td>
<td>A version internal to Operis in which its consultants trial the changes in later external releases.</td>
</tr>
<tr>
<td>4.00.021</td>
<td>15 April 2010</td>
<td>&quot;Rough cut&quot; release of OAK 4.1 and 4.2 features</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Summarize, Map and Compare modified to be more memory efficient in Excel 2007 and 2010.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Added multiple workbook selection to Remove Compare Modifications, Remove Color Formatting and Unhide Cells features.</td>
</tr>
<tr>
<td>4.00.020</td>
<td></td>
<td>A version internal to Operis in which its consultants trial the changes in later external releases.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Fixed pruning of formulas in a single merged cell.</td>
</tr>
<tr>
<td>4.00.019</td>
<td>1 April 2010</td>
<td>&quot;Rough cut&quot; release of OAK 4.1 and 4.2 features</td>
</tr>
<tr>
<td>4.00.016-018</td>
<td></td>
<td>Versions internal to Operis in which its consultants trial the changes in later external releases.</td>
</tr>
<tr>
<td>Version</td>
<td>Date</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| 4.00.015  | 18 March 2010 | Unpublished external release based on 4.00.009 for selected users  
  * Fixed problem with licensing on Win x64 computers. |
| 4.00.014  | 28 January 2010 | A version based on 4.00.000 for customers who hold site licenses for OAK. |
| 4.00.009-013 |          | Versions internal to Operis in which its consultants trial the changes in later external releases. |
| 4.00.008  | 11 August 2009 | A version based on 4.00.009 for companies who hold site licenses for OAK. |
| 4.00.007  | 10 August 2009 | Help updated to 4.00.007                                                     |
| 4.00.006  | 8 July 2009  | A version based on 4.00.005 for companies who hold site licenses for OAK. |
| 4.00.005  |            | A version internal to Operis in which its consultants trial the changes in later external releases.  
  * Improved handling of too many cell formats condition.  
  * Checks that Deapply names is not obstructed by some or all worksheets being protected before it embarks on this sometimes time consuming action rather than announcing the obstacle after it has done most of the work. |
| 4.00.004  | 15 June 2009  | A version released for Operis training courses.  
  Implements an important bugfix: alignment option was intermittently not working when comparing large (Excel 2007) worksheets  
  Also  
  * Search selection dialog didn't remember setting of Match case check box between invocations  
  * Reconstructor dialog defaults didn't match what is described in help. |
4.00.003  A version internal to Operis which trials various bugfixes. Specifically

- Conditional search dialog showed Value2 field when displayed a second time after looking for specified cell value.
- Discrepancy analyzer crashed if reference to unopened workbook was by name rather than reference.
- Reconstructed formula showed discrepancy compared with original in a particular instance.
- =0-Something (or any binary operation involving a constant) blocked confine to additions in reconstructor.
- Pruner picked wrong branch of an IF in a particular instance.
- Option to prune in discrepancy analyser resulted in crash when encountering COUNTIF(..) in a particular instance.
- Pruner crashed in two particular instances.
- Pruner crashed where INDIRECT(..) mentions string expression rather than reference.

Recycle names is much faster in Excel 2007 as OAK now delegates the work to the renaming functionality now available in that version of Excel (so long as the user chooses the Workspace scope; narrower scopes continue to be done formula by formula by OAK)

4.00.002  21 April 2009  A version based on 4.00.000 for customers who hold site licenses for OAK.

Also fixes some bugs

- Exception logging was less informative than intended
- Deapply names crashed if worksheet is protected
- Comparison crashed when comparing a worksheet in compatibility mode with one in large (Excel 2007) mode.

4.00.001  A version internal to Operis which is identical to 4.00.000 except for the inclusion of certain prototype features undergoing trial.
Compared with earlier versions, OAK 4

- has been rewritten in C#, which makes it very much faster than earlier versions which were written in VBA
- is multithreaded, which means that it is more responsive to user interruption of commands
- when loaded into Excel 2007, implements the new Office ribbon interface (but still offers the old menus in earlier versions of Excel)
- is capable of handling the much larger spreadsheets that are possible in Excel 2007
- implements new technologies which actually interpret Excel formulas, on which are built useful new functions that can reconstruct, prune, simplify, dename, and analyse discrepancies between formulas, all of which are useful in checking spreadsheets, particularly through the development of parallel reconstructions of calculations
- exposes its entire interface to COM, meaning that it can be scripted using a programming language such as VBA.

### 9.2 Version history for this help

VERSION 4.40.017
Added section for OAK Registry Settings.

VERSION 4.40.003
Removed references to different installation for 32-bit and 64-bit editions of Excel.
Updated serial number exchange details.

VERSION 4.40.000
Updated to contemplate OAK Essentials tools.
Removed Excel pre-2007 content.
Updated prerequisite information, adding information about the .NET Framework 4.5.2, and Windows version requirements, and removed obsolete prerequisites.

VERSION 4.30.034

Updated documentation for compare ranges and IOAKAPI.CompareRanges to accommodate the R1C1-style formula comparison.

VERSION 4.30.030

Added references to Excel 2016, Windows 10 and the .NET Framework 3.5

Embellished email/phone activation documentation.

VERSION 4.30.013

Added page for problem solving regarding COM registrations.

VERSION 4.30.009


Updated email/phone activation documentation

Updated Operis financial modelling book reference to refer to 3rd edition

VERSION 4.30.006

Added VBA function to generate color values for Compare functions

VERSION 4.30.000

Updated version history for 4.30.000 release

Added section on spreadsheet quality.

Fixed incorrect link in documentation for IOAKAPI.FormatAddress

Fix for IOAKAddIn.CopyAddress which showed CopyLiteral instead

Added page for RenameErrorType and linked to page for IRenameFailure

Added to description of Delink option of IOAKAPI.Reformulate

Updated pruning rules, and API documentation for PruningMode, IOAKAPI.Prune and UI screenshots to include more functions that are now pruned.

Updated figures for Optimize with Windows 8 / Excel 2013 images.
Updated figures for the Compare tool to show the new report columns
Added documentation for the new advanced compare alignment and report change type.
Added comment about ignoring constants in IOAKAPI.DeapplyNames.
Updated figures for deapply names and build names database to include new options.
Updated scripting example to use Windows Vista/7/8-era documents folders
Added instructions for distinguishing between 32 and 64 bit Excel

Updates to accommodate changes in the user interface of Excel 2013 from Excel 2010.
Updates to installation guides for change in naming convention of OAK installers.

Added documentation for the Desktop Edition

Updated licensing documentation to include new server and email-based activation and transfer workflows.

Changed references to "hardwired constants" to "hardcoded constants" in API documentation.

VERSION 4.20.051
Updated version history for 4.20.051 release.

VERSION 4.20.050
Updated version history for 4.20.050 release.
Added references to Excel/Office 2013 and Windows 8
Added instructions for installing .NET 2 in Windows 8

VERSION 4.20.038
Updated version history for 4.20.038 release.

VERSION 4.20.022
Updated version history for 4.20.022 release.

VERSION 4.20.009
Updated version history for 4.20.009 release.
VERSION 4.20.008
Updated some images for new Operis branding colors.

VERSION 4.20.000
Widespread updates to account for changes due to the shift from OAK 4.1 to 4.2.

VERSION 4.10.000
Released 12 Jun 2010 with OAK 4.10.000

VERSION 4.00.023
Widespread updates to account for changes due to the shift from OAK 4.0 to 4.1.

VERSION 4.00.008 TO 4.00.022
Version numbers not used.

VERSION 4.00.007
Released 10 August 2009 with OAK 4.00.007

Description of how to download OAK changed to reflect abolition of difference between installer for Vista and other versions of Windows.

Reference to Eusprig 2009 paper on Excel names added at end of names tutorial.

Description of VBA method RedefineName, previously badly mangled, corrected.

Description of VBA interface IRenameFailure extended to itemise members RenameErrorType enumeration.

Doubly prefixed references added to known limitations.

Format limits added to known limitations.

Material on forcing preinstallation testing moved to section on problem solving.

Version history section added

Numerous other small corrections of spelling, grammar and internal consistency.

Numerous other small corrections to align the help with how OAK actually behaves.
VERSION 4.00.001 TO 4.00.006

Version numbers not used.

VERSION 4.00.000

Released 8 April 2009 with OAK 4.00.000.
Index

A
Accelerator key 57
accounting firms 29
Add-Ins Manager 35
algebraic rules 107
algebraic simplification 228
Analyze discrepancies (OAK command) 109
Apply names (Excel command) 194
Apply names (OAK command) 93
ApplyNames (OAK COM interface) 273
array 113, 130, 133, 183
array formula 130, 133, 183
arrays 125
assumptions book, checking 30
audit methods 29
audit opinion 29
auditing 29

B
blank cells 123, 322
braces {} 183
bugs 339
Build names database (OAK command) 30, 101, 242
BuildNameDatabase (OAK COM interface) 274

C
checking a spreadsheet 28
CHOOSE 220, 221, 230, 253
Cluster Seven 158
code inspection 29, 30
column matching 195, 203, 205, 206
COM 158
Compare (OAK COM interface) 276
Compare workbooks / worksheets (OAK command) 67
CompareWithColors (OAK COM interface) 282
comparing spreadsheets 30, 67, 84
compliance 17
conditional search 121
constant formula cells 119
coordinate adjustment 138, 193
Copy literal (OAK command) 138
CopyLiteral (OAK COM interface) 284
COUNTIF 220, 221, 230, 253
Create names (Excel command) 198
Crystal Ball 338
curly brackets {} 183

D
data book, checking 30
data table 130, 133
Deapply names (OAK command) 94, 261
DeapplyNames (OAK COM interface) 285
Decisioneering 338
defining names 188
Delete names in cells (OAK command) 98
Delete rows / columns (OAK command) 133, 182
DeleteNamesInCells (OAK COM interface) 286
DeleteRowsOrColumns (OAK COM interface) 287
directory in which OAK is installed 22
discrepancy analysis
contrasted with reconstruction 260
introduction 226, 228, 229, 230
distinct formulae 30

e
error 116
error messages
An error has occured in OAK 334
error rates 352
European Spreadsheet Risks Interest Group 352
Eusprig 352
Expression pruning 220, 221

F
F5 187, 188
fan-out 228, 230
finding cell precedents 234, 236
formal model audit 23, 28, 29
Formula | Optimize 216
formula complexity 125

G
going a spreadsheet checked 23, 28, 29
GetVersion (OAK COM interface) 290
Goto (Excel command) 187, 188

H
hardwired constants 118, 319
Hawaii, University of 352
HLOOKUP 220, 221, 230, 253
human error 352
Hyperion 338

I
ICompareResult (OAK COM interface) 290
IDeapplyNamesResult 285
IDeapplyNamesResult (OAK COM interface) 291
IF 220, 221, 230, 253
INDEX 220, 221, 230, 253
INDIRECT 220, 221, 230, 253
Insert Name Paste (Excel command) 191
Insert rows / columns (OAK command) 130, 182
InsertRowsOrColumns (OAK COM interface) 292
installing OAK 34
inventory 163
IOAKAddIn (OAK COM interface) 293
IOAKAPI (OAK COM interface) 295
IRenameFailure (OAK COM interface) 297
IRenameResult (OAK COM interface) 297
J
Jonathan Swan 143
L
Localize names in cells (OAK command) 100
LocalizeNamesInCells (OAK COM interface) 297
LOOKUP 220, 221, 230, 253
Lotus 1-2-3 180
M
Map (OAK COM interface) 299
Map workbook / worksheets (OAK command) 60
MapComponents (OAK COM interface) 301
MapOutput (OAK COM interface) 302
MAX 253
merged cells 115, 125, 130, 133, 182
Microsoft's Excel blog 352
MIN 253
model documentation, checking 30
model inputs, identifying 30
multi area selection 134
N
names
applying 194
arguments for and against using 211
benefits 211
column matching 203, 205, 206
creating 198
deapplying 94
defining 188
drawbacks 211
fine print 208
for constants 207
overview 186
pasting 191
removing 30, 94, 261
row matching 203, 205, 206
tutorial 187
using in formulas 191
NET (.NET framework) 158
version requirements 34
O
OAK
limitations 334
technical requirements 34
what's new 18
OAK COM interface
ApplyNames 273
BuildNameDatabase 274
Compare 276
CompareWithColors 282
CopyLiteral 284
DeapplyNames 285
DeleteNamesInCells 286
DeleteRowsOrColumns 287
GetVersion 290
ICompareResult 290
IDeapplyNamesResult 291
InsertRowsOrColumns 292
IOAKAddIn 293
IOAKAPI 295
IRenameFailure 297
IRenameResult 297
LocalizeNamesInCells 297
Map 299
MapComponents 301
MapOutput 302
OAKResult 302
Operis_OAK 304
Optimize 304
Prune 305
PruningMode 307
Reconstruct 308
RecreateNames 311
RedefineName 312
Reformulate 309
RemoveColorFormatting 313
RemoveHashRefNames 314
SearchFor 315
SearchForArrays 316
SearchForConstantFormula 317
SearchForHardwiredConstant 319
SearchForMerged 320
SearchForPrimaryError 321
SearchForReferenceToBlankCells 322
SearchForUnreferencedCells 324
SearchForValue 325
## Index

**OAK COM interface**
- Summarize 327
- Transpose 328
- UndoCompareModifications 329
- UnhideCells 330

**OAK commands**
- Analyze discrepancies 109
- Apply names 93
- Build names database 30, 101
- Compare workbooks / worksheets 67
- Copy literal 138
- Deapply name 261
- Deapply names 94
- Delete names in cells 98
- Delete rows / columns 133, 182
- Insert rows / columns 130, 182
- Localize names in cells 100
- Map workbook / worksheets 60
- Operis financial modelling book 143
- Optimize 107, 225
- Options 152
- Prune inactive path 108, 225, 253
- Reconstruct 104
- Recreate names 91
- Redefine names 87
- Remove #REF! names 97
- Remove color formatting 141
- Remove compare modifications 84
- Search selection 112, 113, 115, 116, 118, 119, 120, 121, 123
- Summarize workbook 30, 125
- Transpose 134
- Unhide rows and columns 142

**OAK object reference**
- 271, 272
- OAKResult (OAK COM interface) 302

**OFFSET**
- 220, 221, 230, 253

**One-Cell Rule**
- 181

**Operis**
- 29

**Operis financial modelling book (OAK command)**
- 143

**Operis_OAK (OAK COM interface)**
- 304

**Optimize (OAK COM interface)**
- 304

**Optimize (OAK command)**
- 107, 225

**Options (OAK command)**
- 152

**Oracle**
- 338

**Paste Special Link (Excel command)**
- 138

**pasting names into formulas**
- 191

**pivot table**
- 130, 133

**Preinstallation testing**
- 338

**preinstallation tests**
- 339

**primary error cells**
- 116

**Prodiance**
- 158

**programming OAK**
- 158

**Prune (OAK COM interface)**
- 305

**Prune inactive path (OAK commands)**
- 108

**pruning**
- 220, 221, 225, 230, 253

**PruningMode (OAK COM interface)**
- 307

**R**

**Reconstruct (OAK COM interface)**
- 308

**Reconstruct (OAK command)**
- 104, 234, 236, 237, 239, 240, 241, 242, 243, 245

**reconstruction**
- column matching 239
- context 237, 240
- contrasted with discrepancy analysis 260
- formula too long 258
- frozen rows and columns 241
- introduction 234
- left-right consistency 247
- multiple worksheets 236
- names 242
- precedent substitution 243, 245, 246
- pruning 253
- simplification 252
- suitable selections 257
- SUM 255
- timeline 240
- uses 259

**Recreate names (OAK command)**
- 91

**RecreateNames (OAK COM interface)**
- 311

**Redefine names (OAK command)**
- 87

**RedefineName (OAK COM interface)**
- 312

**Reformulate (OAK COM interface)**
- 309

**Remove color formatting (OAK command)**
- 141

**Remove compare modifications (OAK command)**
- 84

**RemoveColorFormatting (OAK COM interface)**
- 313

**RemoveHashRefNames (OAK COM interface)**
- 314

**removing names**
- 30

**row matching**
- 203, 205, 206

**S**

**Sarbanes Oxley**
- 17, 158, 163

**scripting OAK**
- getting started 158
- introduction 158

---

- 377 -

© 1997-2019 Operis Group plc
scripting OAK
  performance 160
Search selection (OAK command) 112, 113
SearchFor (OAK COM interface) 315
SearchForArrays (OAK COM interface) 316
SearchForConstantFormula (OAK COM interface) 317
SearchForHardwiredConstant (OAK COM interface) 319
SearchForMerged (OAK COM interface) 320
SearchForPrimaryError (OAK COM interface) 321
SearchForReferenceToBlankCells (OAK COM interface) 322
SearchForUnreferencedCells (OAK COM interface) 324
SearchForValue (OAK COM interface) 325
simplifying 225
spreadsheet
  comparison 30
  error rates 352
  inventory 163
  research 352
  resources 352
spreadsheet statistics 125
structured references 334
SUMIF 253
Summarize (OAK COM interface) 327
Summarize workbook (OAK command) 29, 30
T
template model 29, 30
Transpose (OAK COM interface) 328
Transpose (OAK command) 134
U
UndoCompareModifications (OAK COM interface) 329
Unhide rows and columns (OAK command) 142
UnhideCells (OAK COM interface) 330
unique formula count 125
University of Hawaii 352
unreferenced cells 120, 324
used range 125, 180
V
VBA 158
Visual Basic 158
VLOOKUP 220, 221, 230, 253
W
workbook summary 180