OpenEdge Development:
Mobile Applications
Third party acknowledgements — See the table of contents for the "Third Party Acknowledgements" appendix.
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preface</td>
<td>11</td>
</tr>
<tr>
<td>1. OpenEdge Mobile Overview</td>
<td>21</td>
</tr>
<tr>
<td>Run-time architecture</td>
<td>22</td>
</tr>
<tr>
<td>Supported Mobile App types</td>
<td>22</td>
</tr>
<tr>
<td>Client access to the AppServer using Mobile services</td>
<td>24</td>
</tr>
<tr>
<td>Access to AppServer classes and procedures using JavaScript objects</td>
<td>25</td>
</tr>
<tr>
<td>Access to Mobile services with user login sessions</td>
<td>28</td>
</tr>
<tr>
<td>Development architecture and tools</td>
<td>30</td>
</tr>
<tr>
<td>Creating Mobile projects</td>
<td>31</td>
</tr>
<tr>
<td>Building Mobile services</td>
<td>31</td>
</tr>
<tr>
<td>Publishing Mobile services</td>
<td>32</td>
</tr>
<tr>
<td>Building Mobile Apps</td>
<td>33</td>
</tr>
<tr>
<td>Building Mobile Apps without the Mobile App Builder</td>
<td>33</td>
</tr>
<tr>
<td>Deployment options</td>
<td>34</td>
</tr>
<tr>
<td>2. Example: A Simple Mobile App</td>
<td>35</td>
</tr>
<tr>
<td>Setting preferences</td>
<td>36</td>
</tr>
<tr>
<td>Creating a new Mobile OpenEdge project</td>
<td>37</td>
</tr>
<tr>
<td>Connecting to the OpenEdge Database</td>
<td>38</td>
</tr>
<tr>
<td>Creating an include file</td>
<td>39</td>
</tr>
<tr>
<td>Creating a new Business Entity</td>
<td>40</td>
</tr>
<tr>
<td>Creating the Mobile App</td>
<td>42</td>
</tr>
<tr>
<td>Adding a JSDO Service to the client</td>
<td>44</td>
</tr>
<tr>
<td>Building and testing the Mobile App</td>
<td>47</td>
</tr>
<tr>
<td>Troubleshooting</td>
<td>48</td>
</tr>
<tr>
<td>Developer Studio</td>
<td>48</td>
</tr>
<tr>
<td>Mobile App Builder</td>
<td>48</td>
</tr>
<tr>
<td>Web Browser</td>
<td>49</td>
</tr>
<tr>
<td>General</td>
<td>49</td>
</tr>
</tbody>
</table>
3. Creating Mobile Services ................................................................. 51
   Getting started with OpenEdge Mobile development ...................... 52
   Starting up Developer Studio for Mobile development .................. 52
   Configuring and managing OpenEdge servers from Developer Studio ... 53
   Creating an OpenEdge Mobile project ......................................... 55
   Creating and testing a Mobile service ....................................... 57
   Coding AppServer services for OpenEdge Mobile .......................... 58
   Singleton classes and procedures as Mobile resources .................. 58
   Coding ABL routines to implement a Mobile resource .................... 59
   Creating Mobile resources ....................................................... 66
   Using the New Business Entity wizard ...................................... 66
   Using the Define Service Interface wizard .................................. 67
   Creating Mobile services ......................................................... 68
   Using the New Mobile Service wizard ....................................... 68
   Modifying and deleting Mobile services .................................... 68
   Publishing Mobile services for testing ..................................... 69
   URIs for accessing Mobile Web applications, services, and resources ... 70
   Mobile URIs for Mobile App access .......................................... 70
   Mobile URIs for testing access from REST clients ....................... 72
   Using a REST client to invoke a Mobile operation ....................... 74

4. Creating Mobile Apps using JSDOs. ................................................. 75
   Creating a Mobile App from Developer Studio ............................. 77
   Getting started with the Mobile App Builder .............................. 78
   JSDO overview ................................................................. 79
      Supporting OpenEdge classes and objects ............................... 79
      How a JSDO maps to a Mobile resource .................................. 80
      How JSDO local storage works ............................................ 81
      Methods of a JSDO and the JSRecord object ............................ 84
      Asynchronous and synchronous execution ............................... 87
      Properties of a JSDO ....................................................... 91
      Requirements for using a JSDO .......................................... 92
   Creating a JSDO for a Mobile resource .................................... 94
   Accessing built-in CRUD operations ...................................... 95
      Read operation ............................................................. 95
      Create operation ......................................................... 97
      Update operation ......................................................... 98
      Delete operation ......................................................... 100
      Calling saveChanges( ) for multiple CRUD operations ............... 103
   Accessing non-built-in invoke operations ................................ 104
      Asynchronous vs. synchronous method execution ...................... 104
      Invocation method example ............................................. 104
   Managing user login sessions ............................................... 106
      Requirements for creating a user login session ....................... 106
      Default Web pages to support Mobile App login ...................... 108
   Getting started with other HTML coding tools ......................... 111
      Using the UIHelper class ............................................... 111
      Using a custom template ................................................. 114
   Publishing Mobile Apps for testing ....................................... 116

5. Using JSDO Services in the Mobile App Builder .............................. 117
   Setting up the sample Mobile service .................................... 118
   Creating the JSDO Services for the Mobile service .................... 121
      Loading the JSDO catalog for the Mobile service .................... 121
      Selecting the Mobile resource from the JSDO catalog ............... 121
      Updating the JSDO Service settings for the resource ............... 122
6. Deploying Mobile Applications ............................................. 165
   Deployment overview .................................................. 166
   Packaging and deploying Mobile services .......................... 167
   Packaging and Deploying iOS Apps .................................. 168
   Packaging iOS Apps .................................................... 168
   Deploying iOS Apps .................................................... 170
   Packaging and Deploying Android Apps ............................. 172
   Packaging Android Apps .............................................. 172
   Deploying Android Apps .............................................. 174
   Packaging and Deploying Mobile Web Apps ......................... 175
   Packaging Mobile Web Apps ........................................... 175
   Deploying Mobile Web Apps .......................................... 176
   Security considerations ............................................... 177
   Web server authentication models .................................. 177
   SSL connections ....................................................... 181
   Mobile device security ............................................... 181

A. ABL to JavaScript Data Type Mapping ................................. 183
   JavaScript data type overview ....................................... 184
   Data type mapping between JavaScript and ABL .................... 186

B. OpenEdge JavaScript Class and Object Reference .................... 189
   JSRecord object ...................................................... 190
   progress.data.JSDO class ............................................ 191
   progress.data.Session class ......................................... 200
   progress.ui.UIHelper class ......................................... 203
   request object ........................................................ 207

C. OpenEdge JavaScript Properties, Methods, and Events Reference .... 209
   add( ) method .......................................................... 212
   addCatalog( ) method ............................................... 214
   addItem( ) method .................................................... 216
   addRecords( ) method ............................................... 217
   afterCreate event .................................................... 221
   afterDelete event .................................................... 223
   afterFill event ........................................................ 225
   afterInvoke event .................................................... 226
   afterSaveChanges event .............................................. 227
   afterUpdate event .................................................... 229
   assign( ) method (JSDO class) ...................................... 231
<table>
<thead>
<tr>
<th>Method/Property</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>assign( ) method (UIHelper class)</td>
<td>233</td>
</tr>
<tr>
<td>async property</td>
<td>234</td>
</tr>
<tr>
<td>authenticationModel property</td>
<td>235</td>
</tr>
<tr>
<td>autoSort property</td>
<td>236</td>
</tr>
<tr>
<td>batch property</td>
<td>239</td>
</tr>
<tr>
<td>beforeCreate event</td>
<td>240</td>
</tr>
<tr>
<td>beforeDelete event</td>
<td>241</td>
</tr>
<tr>
<td>beforeFill event</td>
<td>242</td>
</tr>
<tr>
<td>beforeInvoke event</td>
<td>243</td>
</tr>
<tr>
<td>beforeSaveChanges event</td>
<td>244</td>
</tr>
<tr>
<td>beforeUpdate event</td>
<td>245</td>
</tr>
<tr>
<td>caseSensitive property</td>
<td>246</td>
</tr>
<tr>
<td>catalogURIs property</td>
<td>248</td>
</tr>
<tr>
<td>clearItems( ) method</td>
<td>249</td>
</tr>
<tr>
<td>clientContextId property</td>
<td>250</td>
</tr>
<tr>
<td>data property</td>
<td>251</td>
</tr>
<tr>
<td>display( ) method</td>
<td>252</td>
</tr>
<tr>
<td>fill( ) method</td>
<td>253</td>
</tr>
<tr>
<td>find( ) method</td>
<td>255</td>
</tr>
<tr>
<td>findById( ) method</td>
<td>257</td>
</tr>
<tr>
<td>fnName property</td>
<td>259</td>
</tr>
<tr>
<td>foreach( ) method</td>
<td>260</td>
</tr>
<tr>
<td>getData( ) method</td>
<td>262</td>
</tr>
<tr>
<td>getFormFields( ) method</td>
<td>263</td>
</tr>
<tr>
<td>getFormRecord( ) method</td>
<td>264</td>
</tr>
<tr>
<td>getId( ) method</td>
<td>265</td>
</tr>
<tr>
<td>getListViewRecord( ) method</td>
<td>266</td>
</tr>
<tr>
<td>getSchema( ) method</td>
<td>267</td>
</tr>
<tr>
<td>invocation method</td>
<td>268</td>
</tr>
<tr>
<td>jsdo property</td>
<td>270</td>
</tr>
<tr>
<td>jsrecord property</td>
<td>271</td>
</tr>
<tr>
<td>lastSessionXHR property</td>
<td>272</td>
</tr>
<tr>
<td>login( ) method</td>
<td>273</td>
</tr>
<tr>
<td>loginHttpStatus property</td>
<td>276</td>
</tr>
<tr>
<td>loginResult property</td>
<td>277</td>
</tr>
<tr>
<td>loginTarget property</td>
<td>278</td>
</tr>
<tr>
<td>logout( ) method</td>
<td>279</td>
</tr>
<tr>
<td>name property</td>
<td>280</td>
</tr>
<tr>
<td>onOpenRequest property</td>
<td>281</td>
</tr>
<tr>
<td>paramObj property</td>
<td>283</td>
</tr>
<tr>
<td>record property</td>
<td>284</td>
</tr>
<tr>
<td>remove( ) method</td>
<td>285</td>
</tr>
<tr>
<td>response property</td>
<td>286</td>
</tr>
<tr>
<td>saveChanges( ) method</td>
<td>287</td>
</tr>
<tr>
<td>services property</td>
<td>289</td>
</tr>
<tr>
<td>serviceURI property</td>
<td>291</td>
</tr>
<tr>
<td>setDetailPage( ) method</td>
<td>292</td>
</tr>
<tr>
<td>setFieldTemplate( ) method</td>
<td>293</td>
</tr>
<tr>
<td>setItemTemplate( ) method</td>
<td>294</td>
</tr>
<tr>
<td>setListView( ) method</td>
<td>295</td>
</tr>
<tr>
<td>setSortFields( ) method</td>
<td>296</td>
</tr>
<tr>
<td>setSortFn( ) method</td>
<td>298</td>
</tr>
<tr>
<td>showListView( ) method</td>
<td>301</td>
</tr>
<tr>
<td>sort( ) method</td>
<td>302</td>
</tr>
<tr>
<td>subscribe( ) method</td>
<td>305</td>
</tr>
<tr>
<td>success property</td>
<td>307</td>
</tr>
<tr>
<td>table reference property (JSDO)</td>
<td>308</td>
</tr>
</tbody>
</table>
D. Third party acknowledgements ........................................... 317
## Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1</td>
<td>OpenEdge Mobile run-time architecture for Native and Web Apps.</td>
<td>22</td>
</tr>
<tr>
<td>Figure 2</td>
<td>Accessing OpenEdge Mobile services and resources.</td>
<td>24</td>
</tr>
<tr>
<td>Figure 3</td>
<td>Accessing AppServer data and business logic from Mobile Apps.</td>
<td>25</td>
</tr>
<tr>
<td>Figure 4</td>
<td>OpenEdge Mobile development architecture and tools.</td>
<td>30</td>
</tr>
<tr>
<td>Figure 5</td>
<td>Sample display using the UIHelper class.</td>
<td>111</td>
</tr>
<tr>
<td>Figure 6</td>
<td>Creating JSDO Services for a Mobile service.</td>
<td>121</td>
</tr>
<tr>
<td>Figure 7</td>
<td>Selecting a Mobile resource for which to create JSDO Services.</td>
<td>122</td>
</tr>
<tr>
<td>Figure 8</td>
<td>JSDO Services created for a selected Mobile resource.</td>
<td>122</td>
</tr>
<tr>
<td>Figure 9</td>
<td>Changing the Settings JSDO Service.</td>
<td>123</td>
</tr>
<tr>
<td>Figure 10</td>
<td>Selecting events for a page.</td>
<td>125</td>
</tr>
<tr>
<td>Figure 11</td>
<td>Adding login code for a new Load event.</td>
<td>126</td>
</tr>
<tr>
<td>Figure 12</td>
<td>Adding the new Load event.</td>
<td>127</td>
</tr>
<tr>
<td>Figure 13</td>
<td>Home page for the simple UI for JSDO Services.</td>
<td>129</td>
</tr>
<tr>
<td>Figure 14</td>
<td>Adding the hidden record ID to the list.</td>
<td>130</td>
</tr>
<tr>
<td>Figure 15</td>
<td>Adding a detail page for a list item.</td>
<td>130</td>
</tr>
<tr>
<td>Figure 16</td>
<td>Detail page for simple app with JSDO Services.</td>
<td>131</td>
</tr>
<tr>
<td>Figure 17</td>
<td>Using the Data tab to add a JSDO Service to a page.</td>
<td>133</td>
</tr>
<tr>
<td>Figure 18</td>
<td>Adding the Instance JSDO Service to a page.</td>
<td>133</td>
</tr>
<tr>
<td>Figure 19</td>
<td>Invoking the Instance JSDO Service on a page Load event.</td>
<td>134</td>
</tr>
<tr>
<td>Figure 20</td>
<td>Verifying the order of events for invoking the Instance JSDO Service.</td>
<td>135</td>
</tr>
<tr>
<td>Figure 21</td>
<td>Adding the Read JSDO Service to a page.</td>
<td>136</td>
</tr>
<tr>
<td>Figure 22</td>
<td>Creating a variable to indicate if the Read Service reads local storage only.</td>
<td>137</td>
</tr>
<tr>
<td>Figure 23</td>
<td>Mapping Read Service request parameters.</td>
<td>138</td>
</tr>
<tr>
<td>Figure 24</td>
<td>Mapping Read Service response fields using the built-in schema labels.</td>
<td>138</td>
</tr>
<tr>
<td>Figure 25</td>
<td>Adding JavaScript to map the record ID for the Read Service.</td>
<td>139</td>
</tr>
<tr>
<td>Figure 26</td>
<td>Mapping Read Service response fields to custom list item labels.</td>
<td>140</td>
</tr>
<tr>
<td>Figure 27</td>
<td>Setting the readLocal parameter to read from the AppServer.</td>
<td>140</td>
</tr>
<tr>
<td>Figure 28</td>
<td>Ensuring that the the readLocal setting event happens in the correct order.</td>
<td>141</td>
</tr>
<tr>
<td>Figure 29</td>
<td>Setting the readLocal parameter to read from JSDO local storage.</td>
<td>141</td>
</tr>
<tr>
<td>Figure 30</td>
<td>Invoking the Read Service on a Page show event.</td>
<td>141</td>
</tr>
<tr>
<td>Figure 31</td>
<td>Setting up a list item to provide its record ID to a Row JSDO Service.</td>
<td>142</td>
</tr>
<tr>
<td>Figure 32</td>
<td>Setting the Click event of a list item to navigate to a detail page.</td>
<td>143</td>
</tr>
<tr>
<td>Figure 33</td>
<td>Adding the Row JSDO Service to a page.</td>
<td>143</td>
</tr>
<tr>
<td>Figure 34</td>
<td>Mapping the record ID for a selected list item to the Row Service id parameter.</td>
<td>143</td>
</tr>
<tr>
<td>Figure 35</td>
<td>Mapping the Row Service record to controls on the page.</td>
<td>144</td>
</tr>
<tr>
<td>Figure 36</td>
<td>Invoking the Row Service on a Page show event.</td>
<td>144</td>
</tr>
<tr>
<td>Figure 37</td>
<td>Adding the Create JSDO Service to a page.</td>
<td>145</td>
</tr>
<tr>
<td>Figure 38</td>
<td>Setting the localOnly parameter for a Create Service request.</td>
<td>145</td>
</tr>
<tr>
<td>Figure 39</td>
<td>Setting the localOnly parameter for the Create Service using JavaScript.</td>
<td>146</td>
</tr>
<tr>
<td>Figure 40</td>
<td>Mapping the record ID for the Create Service.</td>
<td>147</td>
</tr>
<tr>
<td>Figure 41</td>
<td>Navigating to the detail page on a successful Create Service invocation.</td>
<td>147</td>
</tr>
<tr>
<td>Figure 42</td>
<td>Invoking the Create Service on a button Click event.</td>
<td>148</td>
</tr>
<tr>
<td>Figure 43</td>
<td>Adding the Update JSDO Service to a page.</td>
<td>148</td>
</tr>
<tr>
<td>Figure 44</td>
<td>Mapping controls to fields for the Update Service request.</td>
<td>149</td>
</tr>
<tr>
<td>Figure 45</td>
<td>Mapping fields to controls for the Update Service response.</td>
<td>150</td>
</tr>
<tr>
<td>Figure 46</td>
<td>Navigating back to the home page after a successful Update Service request.</td>
<td>150</td>
</tr>
<tr>
<td>Figure 47</td>
<td>Invoking the Update Service on a button Click event.</td>
<td>151</td>
</tr>
<tr>
<td>Figure 48</td>
<td>Adding the Delete Service to a page.</td>
<td>151</td>
</tr>
<tr>
<td>Figure 49</td>
<td>Mapping the record ID for the Delete Service.</td>
<td>152</td>
</tr>
<tr>
<td>Figure 50</td>
<td>Confirming the invoke of the Delete Service.</td>
<td>152</td>
</tr>
<tr>
<td>Figure 51</td>
<td>Invoking the Delete Service on a button Click event.</td>
<td>153</td>
</tr>
<tr>
<td>Figure 52</td>
<td>Updating an Invoke Service definition—the Request tab.</td>
<td>155</td>
</tr>
<tr>
<td>Figure 53</td>
<td>Updating an Invoke Service Response tab with a temp-table</td>
<td>156</td>
</tr>
<tr>
<td>Figure 54</td>
<td>Updating an Invoke Service Response tab with a ProDataSet.</td>
<td>157</td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>------------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>55</td>
<td>Adding data records from an Invoke Service to JSDO local storage</td>
<td>158</td>
</tr>
<tr>
<td>56</td>
<td>Adding an ABL return value and output parameter to the Invoke Service Response tab</td>
<td>159</td>
</tr>
<tr>
<td>57</td>
<td>Mapping an output parameter on an Invoke Service Response tab</td>
<td>159</td>
</tr>
<tr>
<td>58</td>
<td>Sample page showing Invoke Service parameter mapping</td>
<td>160</td>
</tr>
<tr>
<td>59</td>
<td>Mapping the Invoke Service return value parameter</td>
<td>160</td>
</tr>
<tr>
<td>60</td>
<td>Testing an Invoke Service parameter value</td>
<td>161</td>
</tr>
<tr>
<td>61</td>
<td>Sample page showing Invoke Service parameter test result</td>
<td>162</td>
</tr>
<tr>
<td>62</td>
<td>Error handling for JSDO Services</td>
<td>163</td>
</tr>
</tbody>
</table>
Tables

Table 1: Required ABL parameters for built-in Mobile operations ........................................... 26
Table 2: Prescribed ABL parameters for built-in Mobile operations ....................................... 61
Table 3: Mobile Application Terminology .................................................................................. 166
Table 4: JavaScript primitive data types .................................................................................... 184
Table 5: Supported non-standard data types .............................................................................. 184
Table 6: JavaScript complex data types ...................................................................................... 185
Table 7: ABL to JavaScript data type mappings .......................................................................... 186
Table 8: JSRecord object properties ........................................................................................... 190
Table 9: JSRecord object methods ............................................................................................. 190
Table 10: progress.data.JSDO properties ................................................................................... 193
Table 11: progress.data.JSDO class-instance methods ............................................................... 194
Table 12: progress.data.JSDO table-reference methods ............................................................... 195
Table 13: progress.data.JSDO events ......................................................................................... 196
Table 14: Example — Using an OpenEdge JSDO ...................................................................... 197
Table 15: progress.data.Session properties .............................................................................. 201
Table 16: progress.data.Session class-instance methods ............................................................ 202
Table 17: Example — Using the OpenEdge Session class ............................................................. 202
Table 18: progress.ui.UIHelper properties .................................................................................. 203
Table 19: progress.ui.UIHelper class-level methods ................................................................. 203
Table 20: progress.ui.UIHelper table-reference methods ............................................................ 204
Table 21: Example — index.html ............................................................................................... 205
Table 22: Example — customers.js ............................................................................................ 206
Table 23: Request object properties ........................................................................................... 207
Preface

This Preface contains the following sections:

- Purpose
- Audience
- Organization
- Using this manual
- Typographical conventions
- Examples of syntax descriptions
- OpenEdge messages
Purpose

Describes how to build and deploy mobile apps using OpenEdge Mobile, an OpenEdge product that exposes application services running on the AppServer as JavaScript objects running in either a Web or mobile client, and uses OpenEdge REST as the transport between them.

This includes a description of the tools and technologies provided with OpenEdge to:

- Build a complete OpenEdge Mobile application, including both server and client code, from a selected data model on an AppServer using Progress Developer Studio for OpenEdge
- Code ABL classes and procedures on an AppServer to be exposed as customized Mobile resources
- Expose specified AppServer procedures or classes as Mobile resources and deploy them in Mobile services that are provided by and managed in an OpenEdge Mobile Web application
- Code instances of OpenEdge JavaScript classes that allow you to access Mobile resources as JavaScript objects similar to how ABL allows you to access data sources through ProDataSet and temp-table objects; thus hiding the details of the REST transport
- Create Web and mobile client UIs (Mobile Apps) using a visual design tool (Progress OpenEdge Mobile App Builder) that is customized to access OpenEdge Mobile services and bind OpenEdge Mobile data to the UI.

Audience

OpenEdge application partners and end users who want to access their OpenEdge data and business logic from either mobile devices or the Web.

Organization

Chapter 1, “OpenEdge Mobile Overview”

Describes the OpenEdge Mobile run-time and development architectures and an overview of how to use them to build Mobile Apps.

Chapter 2, “Example: A Simple Mobile App”

Walks through the process of creating a simple Mobile App directly from an OpenEdge data model.

Chapter 3, “Creating Mobile Services”

Describes the requirements and tools for building Mobile services, including the AppServer coding and Mobile interface annotations required to define Mobile resources.
Preface

Chapter 4, “Creating Mobile Apps using JSDOs”

Describes how to access Mobile services from JavaScript, including how to build Mobile Apps that access these services.

Chapter 5, “Using JSDO Services in the Mobile App Builder”

Describes how to use JSDO Services to visually create and access JSDOs in the Mobile App Builder and visually map JSDO data to UI components with the need for little or no custom JavaScript code.

Chapter 6, “Deploying Mobile Applications”

Describes options for deploying Mobile services and the Mobile Apps that access them.

Appendix A, “ABL to JavaScript Data Type Mapping”

Describes how OpenEdge Mobile maps ABL to JavaScript data types.

Appendix B, “OpenEdge JavaScript Class and Object Reference”

Provides a reference to the OpenEdge JavaScript classes that enable JavaScript access to Mobile services and resources.

Appendix C, “OpenEdge JavaScript Properties, Methods, and Events Reference”

Provides a reference to the properties and methods of the OpenEdge JavaScript classes.

Appendix D, “Third party acknowledgements”

Lists all third-party acknowledgements.

Using this manual

OpenEdge provides a special purpose programming language for building business applications. In the documentation, the formal name for this language is ABL (Advanced Business Language). With few exceptions, all keywords of the language appear in all UPPERCASE, using a font that is appropriate to the context. All other alphabetic language content appears in mixed case.

For the latest documentation updates see the OpenEdge Product Documentation Overview page on PSNDN: http://communities.progress.com/pcom/docs/DOC-16074.

References to ABL compiler and run-time features

ABL is both a compiled and an interpreted language that executes in a run-time engine. The documentation refers to this run-time engine as the ABL Virtual Machine (AVM). When the documentation refers to ABL source code compilation, it specifies ABL or the compiler as the actor that manages compile-time features of the language. When the documentation refers to run-time behavior in an executing ABL program, it specifies the AVM as the actor that manages the specified run-time behavior in the program.
For example, these sentences refer to the ABL compiler’s allowance for parameter passing and the AVM’s possible response to that parameter passing at run time: “ABL allows you to pass a dynamic temp-table handle as a static temp-table parameter of a method. However, if at run time the passed dynamic temp-table schema does not match the schema of the static temp-table parameter, the AVM raises an error.” The following sentence refers to run-time actions that the AVM can perform using a particular ABL feature: “The ABL socket object handle allows the AVM to connect with other ABL and non-ABL sessions using TCP/IP sockets.”

References to ABL data types

ABL provides built-in data types, built-in class data types, and user-defined class data types. References to built-in data types follow these rules:

- Like most other keywords, references to specific built-in data types appear in all **UPPERCASE**, using a font that is appropriate to the context. No uppercase reference ever includes or implies any data type other than itself.
- Wherever `integer` appears, this is a reference to the `INTEGER` or `INT64` data type.
- Wherever `character` appears, this is a reference to the `CHARACTER`, `LONGCHAR`, or `CLOB` data type.
- Wherever `decimal` appears, this is a reference to the `DECIMAL` data type.
- Wherever `numeric` appears, this is a reference to the `INTEGER`, `INT64`, or `DECIMAL` data type.

References to built-in class data types appear in mixed case with initial caps, for example, `Progress.Lang.Object`. References to user-defined class data types appear in mixed case, as specified for a given application example.

References to JavaScript classes and data types

References to JavaScript classes and data types follow common conventions, with JavaScript code in the same fixed-width font used for ABL code.

Typographical conventions

This manual uses the following typographical conventions:

<table>
<thead>
<tr>
<th>Convention</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Bold</strong></td>
<td>Bold typeface indicates commands or characters the user types, provides emphasis, or the names of user interface elements.</td>
</tr>
<tr>
<td><em>Italic</em></td>
<td>Italic typeface indicates the title of a document, or signifies new terms.</td>
</tr>
</tbody>
</table>
## Convention

<table>
<thead>
<tr>
<th>Convention</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMALL, BOLD CAPITAL LETTERS</td>
<td>Small, bold capital letters indicate OpenEdge key functions and generic keyboard keys; for example, GET and CTRL.</td>
</tr>
<tr>
<td>KEY1+KEY2</td>
<td>A plus sign between key names indicates a simultaneous key sequence: you press and hold down the first key while pressing the second key. For example, CTRL+X.</td>
</tr>
<tr>
<td>KEY1 KEY2</td>
<td>A space between key names indicates a sequential key sequence: you press and release the first key, then press another key. For example, ESCAPE H.</td>
</tr>
</tbody>
</table>

### Syntax:

<table>
<thead>
<tr>
<th>Fixed width</th>
<th>A fixed-width font is used in syntax, code examples, system output, and filenames.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fixed-width italics</td>
<td>Fixed-width italics indicate variables in syntax.</td>
</tr>
<tr>
<td>Fixed-width bold</td>
<td>Fixed-width bold italic indicates variables in syntax with special emphasis.</td>
</tr>
<tr>
<td>UPPERCASE fixed width</td>
<td>ABL keywords in syntax and code examples are almost always shown in uppercase. Although shown in uppercase, you can type ABL keywords in either uppercase or lowercase in a procedure or class.</td>
</tr>
</tbody>
</table>

- **This icon (three arrows)** introduces a multi-step procedure.
- **This icon (one arrow)** introduces a single-step procedure.

<table>
<thead>
<tr>
<th>Period (.) or colon (:)</th>
<th>All statements except DO, FOR, FUNCTION, PROCEDURE, and REPEAT end with a period. DO, FOR, FUNCTION, PROCEDURE, and REPEAT statements can end with either a period or a colon.</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ ]</td>
<td>Large brackets indicate the items within them are optional.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Small brackets are part of ABL.</td>
</tr>
<tr>
<td>{ }</td>
<td>Large braces indicate the items within them are required. They are used to simplify complex syntax diagrams.</td>
</tr>
<tr>
<td>{}</td>
<td>Small braces are part of ABL. For example, a called external procedure must use braces when referencing arguments passed by a calling procedure.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Ellipses indicate repetition: you can choose one or more of the preceding items.</td>
</tr>
</tbody>
</table>
Examples of syntax descriptions

In this example, ACCUM is a keyword, and aggregate and expression are variables:

**Syntax**

```
ACCUM aggregate expression
```

FOR is one of the statements that can end with either a period or a colon, as in this example:

```
FOR EACH Customer NO-LOCK:
    DISPLAY Customer.Name.
END.
```

In this example, STREAM stream, UNLESS-HIDDEN, and NO-ERROR are optional:

**Syntax**

```
DISPLAY [ STREAM stream ] [ UNLESS-HIDDEN ] [ NO-ERROR ]
```

In this example, the outer (small) brackets are part of the language, and the inner (large) brackets denote an optional item:

**Syntax**

```
INITIAL [ constant [ , constant ] ]
```

A called external procedure must use braces when referencing compile-time arguments passed by a calling procedure, as shown in this example:

**Syntax**

```
{ &argument-name }
```

In this example, EACH, FIRST, and LAST are optional, but you can choose only one of them:

**Syntax**

```
PRESELECT [ EACH | FIRST | LAST ] record-phrase
```

In this example, you must include two expressions, and optionally you can include more. Multiple expressions are separated by commas:

**Syntax**

```
MAXIMUM ( expression , expression [ , expression ] ... )
```
In this example, you must specify MESSAGE and at least one expression or SKIP \[[n]\], and any number of additional expression or SKIP \[\(n\)] is allowed:

Syntax

```
MESSAGE \{ expression | SKIP \[\(n\)] \} ...
```

In this example, you must specify (include-file, then optionally any number of argument or &argument-name = "argument-value", and then terminate with ):

Syntax

```
\{ include-file
  [ argument | &argument-name = "argument-value" ] ...
\}
```

Long syntax descriptions split across lines

Some syntax descriptions are too long to fit on one line. When syntax descriptions are split across multiple lines, groups of optional and groups of required items are kept together in the required order.

In this example, WITH is followed by six optional items:

Syntax

```
WITH \[ ACCUM max-length \] \[ expression DOWN \]
  \[ CENTERED \] \[ n COLUMNS \] \[ SIDE-LABELS \]
  \[ STREAM-IO \]
```

Complex syntax descriptions with both required and optional elements

Some syntax descriptions are too complex to distinguish required and optional elements by bracketing only the optional elements. For such syntax, the descriptions include both braces (for required elements) and brackets (for optional elements).

In this example, ASSIGN requires either one or more field entries or one record. Options available with field or record are grouped with braces and brackets:

Syntax

```
ASSIGN \{ \[ FRAME frame \] \{ field \[ = expression \] \} \\
  \[ WHEN expression \] \} ...
  \| \{ record \[ EXCEPT field ... \] \}
```
OpenEdge messages

OpenEdge displays several types of messages to inform you of routine and unusual occurrences:

- **Execution messages** inform you of errors encountered while OpenEdge is running a procedure; for example, if OpenEdge cannot find a record with a specified index field value.

- **Compile messages** inform you of errors found while OpenEdge is reading and analyzing a procedure before running it; for example, if a procedure references a table name that is not defined in the database.

- **Startup messages** inform you of unusual conditions detected while OpenEdge is getting ready to execute; for example, if you entered an invalid startup parameter.

After displaying a message, OpenEdge proceeds in one of several ways:

- Continues execution, subject to the error-processing actions that you specify or that are assumed as part of the procedure. This is the most common action taken after execution messages.

- Returns to the Procedure Editor, so you can correct an error in a procedure. This is the usual action taken after compiler messages.

- Halts processing of a procedure and returns immediately to the Procedure Editor. This does not happen often.

- Terminates the current session.

OpenEdge messages end with a message number in parentheses. In this example, the message number is 200:

** Unknown table name table. (200)

If you encounter an error that terminates OpenEdge, note the message number before restarting.

Obtaining more information about OpenEdge messages

In Windows platforms, use OpenEdge online help to obtain more information about OpenEdge messages. Many OpenEdge tools include the following Help menu options to provide information about messages:

- Choose **Help → Recent Messages** to display detailed descriptions of the most recent OpenEdge message and all other messages returned in the current session.

- Choose **Help → Messages** and then type the message number to display a description of a specific OpenEdge message.

- In the Procedure Editor, press the **HELP** key or **F1**.
On UNIX platforms, use the OpenEdge `pro` command to start a single-user mode character OpenEdge client session and view a brief description of a message by providing its number.

**To use the pro command to obtain a message description by message number:**

1. Start the Procedure Editor:

   ```
   OpenEdge-install-dir/bin/pro
   ```

2. Press F3 to access the menu bar, then choose Help → Messages.

3. Type the message number and press ENTER. Details about that message number appear.

4. Press F4 to close the message, press F3 to access the Procedure Editor menu, and choose File → Exit.
OpenEdge Mobile Overview

OpenEdge Mobile allows you to build a complete mobile application, including the mobile client UI, with access to AppServer business logic and data that you can deploy on a variety of popular mobile and Web devices and platforms, including:

- Apple iOS devices, such as the iPhone and iPad
- Android devices, including numerous smart phones and tablets made by a variety of manufacturers
- Web browsers running on a wide variety of computers and mobile devices
- Apache Tomcat Web server for REST access to the AppServer

OpenEdge Mobile supports development for two basic types of client UI (*Mobile App)*:

- *Mobile Web Apps* that run in a Web browser on multiple platforms and devices
- *Mobile Native Apps* that are built the same as Mobile Web Apps, but are deployed to run in native device containers using hybrid technology

You can build and test all application components end-to-end, including the AppServer and corresponding Mobile services, the Mobile App, and the Web transport between them, using OpenEdge Mobile tools available through Progress Developer Studio for OpenEdge (Developer Studio). When your application development is complete, you can then deploy the application components as required to supported mobile devices and Web platforms.

The following sections describe OpenEdge Mobile in greater detail:

- Run-time architecture
- Development architecture and tools
- Deployment options
Run-time architecture

The figures in this section provide an increasingly detailed overview of the OpenEdge Mobile run-time architecture, with shaded and colored elements showing the OpenEdge components. Note that these figures describe Mobile applications built with largely OpenEdge Mobile components from end to end, including an OpenEdge REST transport. However, from any Mobile App, you can also access non-OpenEdge REST Web services using appropriate calls directly to available REST resources.

Supported Mobile App types

Figure 1 shows the basic end-to-end run-time architecture for the two Mobile App types supported by OpenEdge Mobile—Mobile Native Apps running directly in the mobile device OS and Mobile Web Apps running in the Web browser of any hardware platform. The difference is in how the HTML and JavaScript are deployed and run on the client. For a Mobile Native App, HTML and JavaScript files are deployed to an Apple iOS or Android App store to be downloaded and installed on the mobile device and run in the native device container. For a Mobile Web App, the HTML and JavaScript files are deployed on a Web server (A) to load and run in a Web browser (B), as with any HTML Web application. However, the UI might be designed to display in the smaller real estate of a Web browser running in a mobile device.

Figure 1: OpenEdge Mobile run-time architecture for Native and Web Apps
As shown in Figure 1, both Mobile App types can use exactly the same Mobile services to access the AppServer. A Mobile service supports a set of Mobile resources, which provide Web access to an ABL application service on the AppServer.

A Mobile interface defines the ABL API for a single Mobile resource, which can be implemented by either a singleton procedure or a singleton class. (For more information on singleton procedures and classes, see the sections on AppServer coding requirements, in Chapter 3, “Creating Mobile Services.”) A Mobile resource provides access to either a single temp-table or a single ProDataSet with one or more temp-tables, using a standard set of built-in operations that you implement on the AppServer to read and modify the data. A Mobile resource can also provide access to additional AppServer routines to operate on other AppServer data, or even to operate on the same temp-table or ProDataSet data as the built-in operations, but in different ways.

An OpenEdge JavaScript data object (JSDO) is an instance of the OpenEdge JavaScript class, `progress.data.JSDO`, that provides client access to the data and operations of a single Mobile resource. The Mobile App calls JavaScript methods on a JSDO to execute the Mobile operations on the AppServer. The data for these operations is serialized between the Mobile App and the Web server as JSON (JavaScript Object Notation) media.

An OpenEdge Mobile Web application provides the transport (REST over HTTP) for communications between any JSDO running in a Mobile App and a Mobile resource that the application supports. A Mobile Web application can be deployed and the REST transport, including application security, can be managed using the same OpenEdge tools for managing any OpenEdge REST Web application. Note that a single Mobile Web application supports access to Mobile resources implemented by a single OpenEdge AppServer (or a single application service with load balancing).
Client access to the AppServer using Mobile services

Figure 2 provides an overview of how an OpenEdge JSDO accesses a Mobile resource, which works the same for a given JSDO regardless of the type of Mobile App.

Figure 2: Accessing OpenEdge Mobile services and resources

Each JSDO created by a Mobile App can access a single Mobile resource, which can be any of the Mobile resources provided by a particular Mobile service of the Mobile Web application. A single Mobile Web application can support one or more Mobile services, and an Apache Tomcat Web server can host multiple Mobile Web applications.

In Figure 2, the Mobile Web application contains two Mobile services, Inventory and OrderEntry. The OrderEntry service contains two Mobile resources, Orders and Customers, which are implemented by the ABL class files Orders.cls and Customer.cls, respectively, and are accessed, respectively, by the Orders and Customers JSDOs on the client.

Note: The breakdown of resources in the OrderEntry service is for illustration only. It implies that the two resources provide a single temp-table each, one with Order records and one with Customer records. In practice, these temp-tables might be related using a ProDataSet that is provided by a single Mobile resource accessed by a single JSDO.

Note that an OpenEdge Mobile Web application installs and runs in an Apache Tomcat Java container similar to an OpenEdge REST Web application, and can be managed using the same OpenEdge-supported tools. Likewise, an OpenEdge Mobile service is very similar to an OpenEdge REST Web service. The difference between them is in how a Mobile service makes resources available to a Mobile App.
An OpenEdge REST Web service identifies each resource as a unique URI to which a REST client sends an explicit HTTP request that ultimately invokes an AppServer routine. The REST Web service then returns the results of the AppServer routine to the REST client as an HTTP response, which the client must interpret according to the requirements of the particular REST Web service.

An OpenEdge Mobile service, instead, presents specified data and routines of an AppServer singleton class or procedure object as data and operations of a single Mobile resource and maps the operations for that resource directly to corresponding JavaScript methods of a single JSDO. The Mobile service does this mapping with the help of a **JSDO catalog file** that is generated for the service when it is created. This JSDO catalog is a JSON file that contains information about the schema of the data and the operations that each Mobile resource in the service supports. Therefore, a Mobile App must load this JSDO catalog for a Mobile service before creating a JSDO to access any Mobile resource that the service provides.

So, instead of a Mobile App having to identify the URI, send an HTTP request, and interpret the HTTP response for a given REST resource (AppServer routine call), it only has to call the appropriate method or methods on the JSDO to execute the corresponding ABL routine in the class or procedure object on the AppServer. All input parameters, output parameters, and other results of ABL routines are exchanged with the JSDO and its JavaScript methods as corresponding properties of JavaScript objects that the Mobile App passes and returns from the JSDO and its methods.

### Access to AppServer classes and procedures using JavaScript objects

*Figure 3* shows, in greater detail, how a Mobile service maps the operations of a single Mobile resource both to the methods of a JSDO in the Mobile App on the client side and to the ABL routines of a singleton object on the AppServer side.

<table>
<thead>
<tr>
<th>Mobile App</th>
<th>Web Server</th>
<th>OpenEdge AppServer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Customers JSDO</td>
<td>Mobile Web Application</td>
<td>Customers.cls (Mobile Interface)</td>
</tr>
<tr>
<td>fill()</td>
<td>OrderEntry Mobile Svc</td>
<td>ReadCustomers()</td>
</tr>
<tr>
<td>remove()</td>
<td></td>
<td>DeleteCustomer()</td>
</tr>
<tr>
<td>saveChanges()</td>
<td></td>
<td>CreateCustomer()</td>
</tr>
<tr>
<td>add()</td>
<td></td>
<td>UpdateCustomer()</td>
</tr>
<tr>
<td>saveChanges()</td>
<td></td>
<td>GetCreditHistory()</td>
</tr>
<tr>
<td>assign()</td>
<td></td>
<td>GetOrderHistory()</td>
</tr>
<tr>
<td>saveChanges()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>getCreditHistory()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>getOrderHistory()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>invoke operation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>invoke operation</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Figure 3: Accessing AppServer data and business logic from Mobile Apps*
In Figure 3, a singleton Customers class object provides ABL methods to implement the operations of the Customers Mobile resource. The figure shows all the operations that are available for a Mobile resource. The standard, built-in Mobile operations are create, read, update, and delete (the familiar CRUD operations) and you can use Developer Studio to annotate ABL routines of the corresponding Mobile interface that implement them. For each built-in Mobile CRUD operation, you annotate only one ABL routine as the implementation, in this case an ABL method. All the ABL routines that implement the built-in Mobile operations must support exactly the same schema in the form of a single temp-table or ProDataSet parameter. In addition, the read operation implementation must provide an initial CHARACTER input parameter named filter (see Table 1). For non-built-in invoke operations, you can annotate any number of ABL routines using any supported ABL signature. (For more information on the supported ABL data types, see Appendix A, “ABL to JavaScript Data Type Mapping.”) However, the ABL routines you annotate as invoke operations cannot already be annotated to implement built-in CRUD operations.

Table 1 shows the required parameter lists for ABL routines (class methods, internal procedures, or user-defined functions) that implement the built-in Mobile CRUD operations.

### Table 1: Required ABL parameters for built-in Mobile operations

<table>
<thead>
<tr>
<th>Built-in operation</th>
<th>Required parameter list for the implementing ABL routine¹</th>
</tr>
</thead>
<tbody>
<tr>
<td>create</td>
<td>INPUT-OUTPUT { TABLE table-name</td>
</tr>
<tr>
<td></td>
<td>TABLE-HANDLE table-hdl</td>
</tr>
<tr>
<td>read</td>
<td>INPUT filter³ AS CHARACTER , OUTPUT { TABLE table-name</td>
</tr>
<tr>
<td></td>
<td>TABLE-HANDLE table-hdl</td>
</tr>
<tr>
<td>update</td>
<td>INPUT-OUTPUT { TABLE table-name</td>
</tr>
<tr>
<td></td>
<td>TABLE-HANDLE table-hdl</td>
</tr>
<tr>
<td>delete</td>
<td>INPUT-OUTPUT { TABLE table-name</td>
</tr>
<tr>
<td></td>
<td>TABLE-HANDLE table-hdl</td>
</tr>
</tbody>
</table>

¹ If the implementing ABL routine is a class method, its return type must be VOID. The return value of any ABL routine is ignored.

² Because all the built-in operations of a Mobile resource must support the same schema, their implementations all must share either a TABLE, TABLE-HANDLE, DATASET, or DATASET-HANDLE parameter with exactly the same temp-table or ProDataSet schema. NOTE: The schema for any ProDataSet (DATASET or DATASET-HANDLE) parameter can have no data-relations defined with the NESTED option.

³ The filter parameter is passed as a query string that is intended to filter the records returned for the temp-table or ProDataSet parameter of a read operation. Your ABL routine can use this value for any purpose, or ignore the parameter entirely. Note that to allow the prescribed mapping to work between the ABL routine for the built-in read operation and the JavaScript method that calls it, you must name this parameter filter in the signature of the ABL routine.

For information on coding and annotating these ABL routines, see Chapter 3, “Creating Mobile Services.”
As shown in Figure 3 for the Customers resource, when you create a JSDO for a given resource, the JSDO provides JavaScript methods that map to each Mobile operation defined for the resource. Every JSDO supports the same set of built-in methods to call the built-in Mobile operations. For example, calling the fill( ) method always executes the Mobile read operation, which in this case is implemented by an ABL ReadCustomers( ) method. This operation returns all the temp-table records provided by the implementing ABL routine and stores them in a JSDO internal data store referred to as JSDO local storage.

JSDO local storage always matches the schema supported by the JSDO resource as defined by the JSDO catalog for the Mobile service. So, if the JSDO resource supports a single temp-table schema, the JSDO local storage stores the records returned from a single temp-table. If the schema is for a single ProDataSet, it stores the records returned for all temp-tables of the ProDataSet, and maintains them, by default, according to any defined data-relations.

The JSDO supports additional built-in methods to update the contents of its local storage. For example, the add( ) method on the JSDO creates a new record in JSDO local storage. Similarly, the assign( ) method updates, and the remove( ) method deletes an existing record in JSDO local storage. When the Mobile App calls the JSDO built-in saveChanges( ) method, it executes the appropriate built-in Mobile create, update, or delete operation, one at a time, for each record that has changed in its local storage. What happens on the AppServer to the record handled by each operation depends entirely on the implementing ABL routine. In Figure 3, the routines that implement these built-in operations on the AppServer are the DeleteCustomer( ), CreateCustomer( ), and UpdateCustomer( ) methods, respectively.

A JSDO always executes the methods called for built-in CRUD operations asynchronously and supports events to which the Mobile App can subscribe functions to handle the results, whether successful or unsuccessful. These results are returned in an OpenEdge-defined object (request object), with properties that depend on the event and operation outcome. So, each ABL routine that implements a built-in operation can return errors and the same (or a modified version of the) record it received depending on how the operation succeeds or fails. The request object returning these results supports a wide variety of success and failure responses that can be defined for a given mobile application. Note that when a built-in operation fails, any changes associated with that operation are reverted in JSDO local storage.

For non-built-in invoke operations, each ABL routine accepts its defined input parameters and returns its defined output parameters and any return value without any effect on JSDO local storage. Each invoke operation allows the parameters and any return value to be passed as required for the implementing ABL routine. Input parameters are passed as properties of an input JavaScript object, and all results, including any output parameters, return value, and errors, are returned in an output request object similar to the object returned for built-in operations.

To execute an invoke operation, the Mobile App calls the appropriate JSDO invocation method, which executes the corresponding ABL routine. By default, an invocation method executes asynchronously, but can execute synchronously by calling it using the same signature with an added Boolean parameter set to false. For example, in Figure 3, a call to the getCreditHistory( ) invocation method on the JSDO executes the GetCreditHistory( ) method on the AppServer. If getCreditHistory( ) is called asynchronously, the resulting JavaScript request object is returned to the Mobile App as a parameter of any function subscribed as a handler for the appropriate JSDO event. If called synchronously, getCreditHistory( ) returns the same JavaScript request object as the method return value.
Chapter 1: OpenEdge Mobile Overview

Access to Mobile services with user login sessions

In Figure 3, it is the JSDO catalog for the OrderEntry service that ultimately enables JavaScript methods of the Customers JSDO to execute ABL methods of the Customers class to implement operations of the Customers resource. Before it can create and use a JSDO to access a Mobile resource, an OpenEdge Mobile App must log into the appropriate Mobile Web application and load the catalog file for the Mobile service that provides the resource.

Depending on the Web server and Mobile Web application security configuration, the Mobile App can log into the Web application as directed by the Web server, native device container, or browser, or by using the login() method of the OpenEdge JavaScript class, progress.data.Session. If the Mobile App login successfully occurs before calling this login() method, the Mobile App must still call the method in order to establish the login session with the Mobile Web application and load its JSDO catalog files.

To use a Session object to establish the login session, the Mobile App creates an instance of the class and invokes the login() method on that instance, passing the Mobile Web application URI and optional user credentials as parameters. The exact user login sequence can be affected by the Web server security configuration, the Mobile App type, design, and platform, and how Mobile Web application resources are protected on the Web server. For example, if the required user credentials have already been authenticated prior to calling this method, any user credentials you pass to the method are ignored. For more information on establishing a login session for a Mobile App, see Chapter 4, “Creating Mobile Apps using JSDOs.”

Loading the JSDO catalog

Once the login() method on a Session object has been successfully called for a given OpenEdge Mobile Web application, the Mobile App can then call the addCatalog() method on the Session object to load the JSDO catalog for each Mobile service in the Mobile Web application it needs to access. Each JSDO catalog has a URI that you pass to this method, along with optional user credentials if the catalog requires separate user authentication. Note that by default the filename for a JSDO catalog has the name of the Mobile service for which the catalog is created. Therefore, its filename and extension have the form, service-name.json. Once a login session is successfully established, and all required JSDO catalogs are loaded, the Mobile App can invoke operations for all authorized Mobile services and resources.

Note that Mobile services and resources in a Mobile Web application are protected using the Spring Security framework, which is installed with OpenEdge to secure Mobile and REST Web applications. For more information on using Spring Security to secure Mobile or REST Web applications, see the sections on managing REST application security in OpenEdge Application Server: Administration.

Single sign-on to the AppServer

OpenEdge also supports single sign-on (SSO) to the AppServer for a user login session, depending on the OpenEdge Mobile Web application security configuration. When appropriately configured, and after a Mobile Web application authenticates a login session, the application creates an SSO client-principal that it sends to the AppServer with each AppServer request generated by a JSDO. This client-principal contains the user ID used to login a Mobile App and a unique session ID. The AppServer can then access this client-principal using the CURRENT-REQUEST-INFO attribute on the SESSION system handle and use it to establish an SSO login session for the current request.
For information on configuring a Mobile Web application to provide an SSO client-principal to the AppServer with each AppServer request, see the sections on SSO support for REST applications in OpenEdge Application Server: Administration. For more information on accessing this SSO client-principal on the AppServer, see the reference entry for the CURRENT-REQUEST-INFO attribute in OpenEdge Development: ABL Reference and the sections on establishing login sessions and client login context in OpenEdge Application Server: Developing AppServer Applications.
Development architecture and tools

Figure 4 shows an overview of the development architecture and tools for OpenEdge Mobile.

The development tool set consists of two main components:

- **Progress Developer Studio for OpenEdge (Developer Studio)** — The main OpenEdge on-premise tool where you initiate all OpenEdge Mobile development work.

- **Progress OpenEdge Mobile App Builder** — Available in the cloud, you can use the Mobile App Builder to visually design and code a complete Mobile App for all supported Web and Native App platforms. This tool provides built-in coding services that allow you to create and access JSDOs in a Mobile App based on Mobile resources that you select, with little or no additional coding. When you are ready to test, you can run the Mobile App in an emulator for the selected platform. If you choose not to use the Mobile App Builder, you can also use other JavaScript and HTML coding tools in Developer Studio to build your Mobile App.

The following sections describe the development tasks that you can perform using this OpenEdge Mobile development platform:

- Creating Mobile projects
- Building Mobile services
- Publishing Mobile services
- Building Mobile Apps
- Building Mobile Apps without the Mobile App Builder
Creating Mobile projects

To build Mobile services and Apps, you must work in a Mobile project, which is an OpenEdge project type that you create in Developer Studio. You can create a Mobile project either as a new project or by adding the required facets to or performing a Mobile function in an existing OpenEdge project. When you first create a Mobile project in Developer Studio, you must also enter Progress user credentials in Developer Studio that you share with the Mobile App Builder. This allows the on-premise Mobile project in Developer Studio and the corresponding Mobile App Builder project in the cloud to exchange files and data.

When you create a Mobile project in Developer Studio, you also have the option of an Express setup that automatically builds, deploys, and launches a complete Express Mobile application based on a single database table that you select. This Express Mobile project thus provides a way to rapidly create a working Mobile application as a starting point that you can extend using all the features of Developer Studio and the Mobile App Builder. The Express Mobile application includes a Mobile App complete with login, list, and detail pages designed to run on a mobile phone or tablet and includes support for all Mobile CRUD operations on the selected table. These Mobile operations are supported by a Mobile service that is automatically built from a default ABL class with methods generated to implement the operations on the table. When you complete the Express setup for the project, the working Mobile App automatically opens in your configured HTML5 browser.

You can make modifications to an Express Mobile application using the Developer Studio and Mobile App Builder features described in this manual and the online help, such as adding new Mobile resources and App pages. You can also use it unchanged as a demo application to rapidly show how a Mobile App can access any OpenEdge data provided on an AppServer.

Building Mobile services

Once you have a Mobile project in Developer Studio, you can build Mobile services. Depending on what you are trying to accomplish, you might begin by defining a database connection for the project and also defining a database connection for an OpenEdge AppServer (outside of Developer Studio) that is associated with an AppServer facet of the project. Also, when you develop the ABL source code for the Mobile resources you include in Mobile services, it is created in a project folder associated with this AppServer.

As noted in the "Run-time architecture" section on page 22, each Mobile resource is created from a Mobile interface that defines the ABL API for that resource in a single class or procedure file. This class or procedure must be coded as a singleton object that a Mobile application creates on the AppServer to run the corresponding resource of a given Mobile service.
You can use Developer Studio wizards to define a Mobile interface in one of the following ways:

- Use the New Business Entity wizard to create a new Business entity class from a data model based on a single temp-table or a ProDataSet with one or more temp-tables. Creating a Business entity class defines a default set of ABL class methods, each of which implements a create, read, update, or delete (CRUD) operation on the specified data model. In addition, you can have a new Business entity class and its methods automatically annotated to define the interface for the built-in operations of a Mobile resource. You then manually code the body of each ABL method to implement the functionality for each operation. You can then directly use this new, annotated Business entity class file to define a single resource in a Mobile service.

- Use the Define Service Interface wizard to annotate the ABL routines of existing class or procedure files based on the corresponding temp-table or ProDataSet data model. This can be a class or procedure file that is already coded as a singleton, including an existing Business entity class that is not already annotated as a Mobile interface. This allows you to define the Mobile interface from existing AppServer code, or to define non-built-in invoke operations, for example, in an existing Business entity class that currently has only built-in CRUD operations defined. In any case, once an existing class or procedure file is appropriately annotated, you can use this file to define a single Mobile resource in a Mobile service.

**Caution:** Developer Studio relies on wizards to annotate all class and procedure files as Mobile interface files in order to correctly define resources for Mobile services. Do not manually enter or modify these annotations in source code, or the Mobile service created with these resources might fail to function.

Once you have coded and annotated all the class and procedure files needed to define the resources of a Mobile service, you can build the service in Developer Studio using the New Mobile Service wizard to select the Mobile resource files to include in the service. Once you complete this wizard, Developer Studio defines the Mobile service along with its JSDO catalog file. (For an overview of JSDO catalog files, see the “Client access to the AppServer using Mobile services” section on page 24.)

For more information on building Mobile services, see Chapter 3, "Creating Mobile Services."

### Publishing Mobile services

In order to create Mobile Apps to access the Mobile services you have built, you can publish the services to a built-in Web server (OE Web Server) that is installed with OpenEdge. The OE Web Server is an Apache Tomcat Web server configured to run both OpenEdge Mobile Web applications and OpenEdge REST Web applications. When you publish Mobile services, you package them in a WAR file that contains a Mobile Web application with one or more Mobile services, including their JSDO catalogs and related artifacts, and any Mobile App to be deployed to the Web server. Developer Studio provides features to package your Mobile services (along with any associated Mobile App) into the WAR file and deploy it to the OE Web Server. For the default Tomcat configuration, this automatically installs the Mobile Web application and its Mobile services, and enables them for access by any authorized Mobile App.
Building Mobile Apps

As previously described for Figure 4, the Progress OpenEdge Mobile App Builder is the main OpenEdge Mobile tool for building Mobile Apps. It is hosted in the cloud and can launch within any external Web browser that supports HTML5 and that you also configure to run the Mobile App Builder from Developer Studio. When you build a Mobile App with this tool, you design and build the UI for the App and code the JavaScript to access the Mobile services and associated JSDO catalog files that you have published to the OE Web Server. All the UI design and coding in the Mobile App Builder creates the HTML and JavaScript files for the Mobile App in a project that is also hosted in the cloud, but shared with your on-site Mobile project in Developer Studio.

The Mobile App Builder allows you to visually design the UI for a Mobile App using drag-and-drop, similar to the Visual Designer in Developer Studio for building .NET UIs. It provides built-in templates for designing HTML pages for a given device type, such as a phone or tablet. It also provides built-in coding services (JSDO Services) that allow you to access JSDOs and map JSDO data to HTML elements based on JSDO catalog files that you load and the Mobile resources that you select for them, using little or no additional JavaScript code. You can optionally code custom JavaScript when adding events for both these JSDO Services and the HTML elements on a page.

When you are ready to test, you can have the Mobile App Builder publish the HTML and JavaScript files for the Mobile App to your on-site Mobile project in Developer Studio, from where it can be published to the OE Web Server. A test function in the Mobile App Builder then launches an emulator based on the type of Mobile App you are building (Mobile Web App, or Mobile Native App for an iPhone or Android device), so you can see the App run as it might appear in the target platform, but displayed in your HTML5 Web browser. For a Mobile Native App, you can also have Mobile App Builder package it into a mobile device deployment file (IPA for Apple iOS or APK for Android), which you can install for testing on the appropriate mobile device.

Building Mobile Apps without the Mobile App Builder

If you prefer, you can build your own Mobile Apps without using the Mobile App Builder. You can use the built-in Developer Studio tools for coding HTML and JavaScript, instead. OpenEdge Mobile provides a progress.ui.UIHelper JavaScript class to help map JSDO data to HTML pages that you build yourself. For a Mobile Web App, you can publish the files for testing and deployment using the OpenEdge Mobile wizards in Developer Studio. For a Mobile Native App, you need, in addition, to have external access to a hybrid packaging tool, such as Adobe PhoneGap, in order to test your Mobile App in the native container of a mobile device.

For more information on building Mobile Apps, see Chapter 4, "Creating Mobile Apps using JSDOs."
Deployment options

For production deployment, the options basically consist of what you can deploy to an Apache Tomcat Web server (preferably an installed OE Web Server) and what you need to deploy to mobile app stores for download and installation on mobile devices, and the security considerations for all of it.

At a minimum, you deploy your Mobile services for a single Mobile application to the Web server, typically all in one Mobile Web application WAR file. If you are deploying a Mobile Web App for access by Web browsers on mobile devices, you can deploy it either in the same WAR file with your Mobile services, if you can use the same Web server, or in a separate Web application WAR file, if you need the Mobile App to load from a different Web server.

For a Mobile Native App, which installs and runs in the native container of a mobile device, you need to package an Apple iOS App in an IPK file and package an Android App in an APK file for deployment to the appropriate app stores. Each type of Mobile Native App has additional deployment requirements that you need to complete in order to make the App available from an app store.

One consideration that affects all Mobile Native Apps and any Mobile Web App that you deploy to a Web server that is different from the Web server where you deploy your Mobile services is the need to use absolute URIs to login and access the JSDO catalogs for your Mobile services. This means that you need to maintain separate JavaScript sources for the different environments where you deploy these Mobile Apps, for example, one source for development testing and another for production deployment.

Security considerations really begin with Mobile App development, since Mobile App design can affect your security options. You can make some of the important security decisions during Mobile App development, which can then be duplicated during deployment the deployment of Mobile Services. Other security options, especially involving user controls, can only be configured during production deployment.

For more information on these deployment options and how to approach them, see Chapter 6, “Deploying Mobile Applications.”
Example: A Simple Mobile App

This chapter steps you through the creation of a simple Mobile application. It describes the creation of a simple Mobile service with access to an OpenEdge database using Progress Developer Studio for OpenEdge. It also describes how to use the Progress OpenEdge Mobile App Builder to create a simple Mobile App (UI) that accesses this service and displays information from the database.

- Setting preferences
- Creating a new Mobile OpenEdge project
- Connecting to the OpenEdge Database
- Creating an include file
- Creating a new Business Entity
- Creating the Mobile App
- Adding a JSDO Service to the client
- Building and testing the Mobile App
- Troubleshooting
Setting preferences

When you first start Progress Developer Studio for OpenEdge (Developer Studio) to create a Mobile App, you need to set some preferences. The Mobile App Builder runs in a Web browser, so you must specify one. The browser must be HTL5 compliant.

Note: The names for the project, Mobile App, Business Entity, and other items given in this example must be used exactly as is for the Mobile App to work. These names are used in the code provided in this chapter. See the “Troubleshooting” section on page 48 for more information.

To select a web browser:

1. Click Window → Preferences → General → Web Browser.
2. Click Use external web browser.
3. Select a supported browser. If no supported browser is listed but you have one installed, click New, type in the browser’s name, and then browse to the installed location.
4. Click Apply.

You must also enter login credentials to use the Mobile App Builder.

To enter your credentials:

1. Click Window → Preferences. Expand Progress OpenEdge and click Mobile App Builder. Click Next.
2. Enter the user name and password associated with your Progress ID account.
3. Click OK.
Creating a new Mobile OpenEdge project

Now that you have selected a browser and entered your credentials, you can start creating the app.

To create a new Mobile project:

1. Select File → New → OpenEdge Project. Name the project *MyMobile* and select Mobile from the drop-down menu. Click Next.

2. Click Next again to accept the defaults on the AVM and layout options page.

3. On the Define AppServer content module page, select *restbroker1* as your server. Click Next.

4. A Mobile service is created automatically. On the Create a Mobile Service page, select *restmgr1* as your server. Click Next.

5. On the Create a Mobile App page, select *restmgr1* as your server. Note that Developer Studio automatically appends “App” to your project name to create the name of your Mobile App. Click Next.

6. Click Next to accept the PROPATH defaults.

7. Click Configure database. For this sample app, add a new connection to the *Sports2000* database provided with OpenEdge. Enter *Sports2000* as the connection name. Use *localhost* as the Host name at port 5555.

8. Click Finish.

9. Click Yes when prompted if you want to open the server perspective.

By default, the Mobile App Builder will now be launched in a browser window. Minimize the browser window for now and return to Developer Studio.
Connecting to the OpenEdge Database

To enable your app to display OpenEdge data, the OpenEdge AppServer must be connected to an OpenEdge database.

To connect the AppServer to the database:

1. Double-click `restbroker1` in the Server view.
2. Click Open launch configuration.
3. Go to the Database tab, and select the Show all radio button.
5. Close the `restbroker1` editor.
6. Right-click `restbroker1`, and click Start in the pop-up menu.
Creating an include file

Before creating a new Business Entity for the Mobile service, you must create an include file that contains a temp table that describes the schema of the Sports2000 database.

To create an Include file:

1. Click File→New→ABL Include.
2. Select a container and name the file dsCustomer.i. Click Finish.
3. In the body of the include file, copy and paste the following code:

```apl
DEFINE TEMP-TABLE eCustomer NO-UNDO BEFORE-TABLE beCustomer
FIELD CustNum AS INTEGER
FIELD Name AS CHARACTER FORMAT "X(20)"
FIELD Address AS CHARACTER
FIELD Phone AS CHARACTER
FIELD SalesRep AS CHARACTER
FIELD Balance AS DECIMAL
FIELD State AS CHAR
FIELD numOrders AS INT
INDEX CustNum IS UNIQUE PRIMARY CustNum
INDEX Name NAME.

DEFINE DATASET dsCustomer
   FOR eCustomer.
```

4. Save the file in your working directory and close the file.
Creating a new Business Entity

A Business Entity is an ABL class (CLS) file that includes pre-defined Mobile service annotations. For more information, see "Coding ABL routines to implement a Mobile resource" section on page 59.

To create a new Business Entity:

1. Click File→ New→ Business Entity.

2. Browse to \MyMobile\AppServer for the Package root, and enter dsCustomer as the Business entity name. Click Next.

3. In the Schema file field, browse to the location of the dsCustomer.i file you created previously. From the schema displayed, select dsCustomer.

4. Click Finish.
5. In the Business entity source code, copy and paste the following code to the `ReaddsCustomer` method. This reads records from the `Sports2000` database.

```plaintext
DEFINE DATA-SOURCE srcCustomer FOR Customer.
   EMPTY TEMP-TABLE eCustomer.


filter = "where custnum < 100".
IF filter NE "" AND filter NE ? THEN

DATASET dsCustomer:FILL().

BUFFER eCustomer:DETACH-DATA-SOURCE().

RETURN.
```

6. Save and close the class file.

To add the Business entity to the Mobile service:

1. In the Project Explorer pane, expand Defined Services and right-click MyMobileService. Click Edit.

2. On the Edit a Mobile Service page, click Next.

3. On the Modify a Mobile Service page, select the check box next to the `dsCustomer.cls` file. Click Finish.
Creating the Mobile App

You are now ready to use the Mobile App Builder to create the Mobile App. By default, the Mobile App Builder creates a **home** page that you can customize. You can open the Mobile App Builder at any time by double-clicking your application name in the **Project Explorer** pane. For more information, see “Getting started with the Mobile App Builder” section on page 78.

**To customize the client UI:**

1. In the **Project** pane, expand the **Pages** folder and click the **home** page.
2. Click **Caption** and change the text to **My Mobile App**.
3. Add a list to the page by dragging and dropping from the **Components** pane. In the **Properties** pane, change the **Name** field to **CustList** and change the **Items** field to **1**.

4. Click the list item itself and change the **Name** field to **CustItem**.
5. Drag and drop a label on top of the list item. Change the name of the label to **CustName**.

**To add an event to the page:**

1. Expand the **Events** panel at the bottom of the designer.
2. In the **Component** menu, select **home**.
3. In the **Event** menu, select **Load**.

4. In the **Action** menu, select **Run JavaScript**.

5. In the JavaScript editor that appears, copy and paste the following code:

```javascript
var settings;
var pdsession;
var cMsg = "ok";

try {
  /* CHANGE THIS TO POINT TO YOUR SETTINGS SERVICE */
  settings = MyMobileService_dsCustomer_Settings;

  pdsession = new progress.data.Session();
  var loginResult = pdsession.login(settings.serviceURI,"","");

  if (loginResult != progress.data.Session.LOGIN_SUCCESS) {
    console.log('ERROR: Login failed with code: ' + loginResult);
    switch (loginResult) {
      case progress.data.Session.LOGIN_AUTHENTICATION_FAILURE:
        cMsg = 'Invalid user-id or password';
        break;
      case progress.data.Session.LOGIN_GENERAL_FAILURE:
        default:
        cMsg = 'Service is unavailable';
        break;
    }
  }
}

catch (e) {
  cMsg = "Failed to log in";
  console.log(e.stack);
}

if (cMsg != "ok") {
  alert(cMsg);
  return;
}

pdsession.addCatalog(settings.catalogURI);
```

6. **Click Add event.**

**Note:** Mobile App Builder provides several events. Some are HTML events available to any Mobile App, and some are native device (PhoneGap) events available only to a Mobile Native App. For example, if you install and run this Mobile App as a Native App, the HTML **load** event initially fires, starting a login session on the server. When launching the Mobile Native App a second time, it is possible that the app is already loaded. In this case, execution resumes and the **load** event does not fire. If the device goes offline, the network and the data will be unavailable. One way to test the network connection and restore any lost login session is to create a PhoneGap **resume** event that executes the same login code as the **load** event when the app resumes execution.

For more information on HTML events, see [http://en.wikipedia.org/wiki/DOM_events](http://en.wikipedia.org/wiki/DOM_events). For more information on PhoneGap events, see [http://docs.phonegap.com/en/1.0.0/phonegap_events_events.md.html](http://docs.phonegap.com/en/1.0.0/phonegap_events_events.md.html). For more information on working with events and actions in Mobile App Builder, see the online Mobile App Builder documentation.
Adding a JSDO Service to the client

An OpenEdge JavaScript data object (JSDO) provides client access to ABL data. A JSDO Service is a coding service in the Mobile App Builder that simplifies the mapping between JSDO data and the HTML elements of a Mobile App. For more information on the JSDO class, see Chapter 4, “Creating Mobile Apps using JSDOs.” For more information on JSDO Services, see Chapter 5, “Using JSDO Services in the Mobile App Builder.”

To add the JSDO Service to the client:

1. In the Project pane, select Create New → Service.
2. Select the JSDO Service radio button.
3. Click Upload a file and browse to the MyMobileService.json file. The file is located in your workspace folder in \MyMobile\WebContent.
4. Click Select resources.
5. Select MyMobileService.ds.Customer and click Create services.
6. Click Close.
8. Change the default value of catalogURI to:

   http://Your-IP-Address:8980/MyMobileService/static/mobile/MyMobileService.json
9. Change the default value of serviceURI to:

   http://Your-IP-Address:8980/MyMobileService

10. Click Save.

11. In the Project pane, click home to get back to the UI. Click the Data tab that appears to the left of the components.

12. In the drop-down menu labeled Add data source, click Service, then click MyMobileService_dsCustomer_JSDO. Click Add and enter Customer for the name of the Service.

13. Add another data source by clicking Service again. Click MyMobileService_dsCustomer_Read. Click Add and enter Read for the name of the Service.

14. Click Edit mapping next to the Read Service.

15. Click the Response tab.

16. Expand dsCustomer and all of its components on the left, and do the same for home and all of its components on the right.

   ![Diagram showing mapping between eCustomer and CustItem]

17. Find Name under eCustomer and drag and drop it onto Text under CustName.

18. Drag and drop eCustomer onto CustItem.

19. Click Back to datasources.

20. Click the Design tab and expand the Events tab at the bottom of the page.
21. Create a new event:

   a. In the Component menu, select home.

   b. In the Event menu, select Load.

   c. In the Action menu, select Invoke Service.

   d. Select Customer for Service.

   e. Click Add event.

   **Note:** In general, JSDO Services (like most Web services) execute asynchronously. This means that when the Customer JSDO Service is invoked as an action of the Load event for this step, the Service might not complete before executing any additional action that you define for the same Load event. If this additional action depends on a successful result of invoking the Customer Service, you must instead define that action for the Success event of the Customer Service rather than as an action of the same Load event that invokes the Service. For example, if the additional action is an invocation of the Read JSDO Service, you can define this action for the Success event of the Customer Service, as shown in Step 23. This ensures that the Read Service is not invoked before the Customer Service completes successfully.

   Note also that the Customer Service in this particular example is an instance initialization JSDO Service that runs synchronously. However, it is a best practice to use the Success event to add a dependent action even for a synchronous Service call like this.

22. Click the Data tab and expand the Events tab at the bottom of the page.

23. Create a new event:

   a. In the Component menu, select Customer.

   b. In the Event menu, select Success.

   c. In the Action menu, select Invoke Service.

   d. Select Read for Service.

   e. Click Add event.

24. Click Save.
Building and testing the Mobile App

The Mobile App is now ready to be tested in the built-in emulator.

To test the app:

1. In the Server view in Developer Studio, right click restmgr1, and click Start in the pop-up menu.
2. Click Run → Run Configurations.
3. Double-click Progress OpenEdge Mobile in the pane on the left.
4. Enter a name for the configuration. Select MyMobile from the Project menu, and MyMobileApp from the Mobile App menu.
5. Select the radio button for Run on server. Developer Studio will automatically generate the necessary link.
6. Click Run. After launching is complete, the mobile preview browser will open, and the Mobile App will run in the emulator.

To view the app on a mobile device, open a browser on your device and enter the following URL:

http://Your-IP-Address:8980/MyMobileApp

For information on deploying the Mobile App for Web, Android, or Apple iOS, see Chapter 6, “Deploying Mobile Applications.”
Troubleshooting

If MyMobileApp does not display data, check the following possible sources of error.

**Developer Studio**

In the **Servers** tab:

1. Make sure restbroker1 is running and synchronized. If not, make sure your database server is running and that restbroker1 is connected to it.

2. Make sure restmgr1 has an entry for the Mobile service and is synchronized. If not, you may need to stop the Tomcat server, delete some artifacts, and then restart the Tomcat server.
   a. In a proenv shell, type `protc stop` and press Enter.
   b. Navigate to `Progress/OpenEdge/servers/tomcat/webapps` in your installation directory. There should be two WAR files named `MyMobileApp.war` and `MyMobileService.war` and two folders named `MyMobileApp` and `MyMobileService`. Delete the WAR files and the folders. If all four artifacts are not there, delete what is there.
   c. In a proenv shell, type `protc start` and press Enter.
   d. In the **Servers** tab, right-click restmgr1 and choose **Publish**.

   **Note:** After republishing several times, there are sometimes leftover files that can be deleted. To delete these files, use the `protc clean` command after `protc stop` and before `protc start`.

**Mobile App Builder**

Check the following in the Mobile App Builder in your Web browser.

1. Open `MyMobileService_dsCustomer_Settings` in the **Services** folder. (If the name of your `_Settings` document has a different name, open this file.) Check the following settings:
   a. The **catalogURI** should be set to:

   ```
   http://Your-IP-Address:8980/MyMobileService/static/mobile/MyMobileService.json
   ```

   If you did not use `MyMobileService` for the service name, replace `MyMobileService` with your Mobile service name.
   b. The **serviceURI** should be set to:

   ```
   http://Your-IP-Address:8980/MyMobileService
   ```
c. The `resourceName` setting should be `dsCustomer`. If you did not use `dsCustomer` for your Business entity name, replace `dsCustomer` with the correct name.

2. Check the JavaScript associated with the `Load` event of the home page.
   
a. The first parameter in the `pdsession.login( )` method call should be `MyMobileService_dsCustomer_Settings.serviceURI`. If your service name is not `MyMobileService` and/or your Business entity is not named `dsCustomer`, modify this parameter accordingly.

b. The first parameter in the `pdsession.addCatalog( )` method call should be `MyMobileService_dsCustomer_Settings.catalogURI`. If your service name is not `MyMobileService` and/or your Business entity is not named `dsCustomer`, modify this parameter accordingly.

c. Check that "Run JavaScript" is the action for the first `Load` event and "Invoke Service" is the action for the second `Load` event.

**Web Browser**

Open a new browser window and try the following.

1. The URL `http://Your-IP-Address:8980` displays the Tomcat home page. If not, open a proenv shell, type `protc start`, and press Enter.

2. The URL `http://Your-IP-Address:8980/MyMobileService/rest` displays the available REST Web services. If you do not have any available REST Web services, you need to republish `MyMobileService`. See the "Developer Studio" section on page 48.

**General**

Open the Windows Firewall and make sure port 8980 is open.
Creating Mobile Services

As you may recall from Chapter 1, “OpenEdge Mobile Overview,” an OpenEdge JavaScript data object (JSDO) in a Mobile App allows you to access the data and operations of a Mobile resource provided by a Mobile service. When you develop an OpenEdge Mobile application, you typically first develop the interface to each Mobile resource that you intend to access using JSDOs, then create the Mobile services to provide these Mobile resources to a Mobile App. If this is your first Mobile application, you need to create a new Mobile project in Progress Developer Studio for OpenEdge (Developer Studio) in order to start OpenEdge Mobile development.

This chapter describes options and procedures for starting Mobile development and creating and testing Mobile services, as follows:

- Getting started with OpenEdge Mobile development
- Coding AppServer services for OpenEdge Mobile
- Creating Mobile resources
- Creating Mobile services
- Publishing Mobile services for testing
- URIs for accessing Mobile Web applications, services, and resources
Getting started with OpenEdge Mobile development

The following sections describe how to get started with Mobile development:

- Starting up Developer Studio for Mobile development
- Configuring and managing OpenEdge servers from Developer Studio
- Creating an OpenEdge Mobile project
- Creating and testing a Mobile service

Starting up Developer Studio for Mobile development

When starting up Mobile development, you might want to begin with a review of the features you need and set some basic options.

To start up Developer Studio for Mobile development:

1. Choose one of the items under Start → Progress → Developer Studio 3.7, preferably using a separate Mobile workspace if you already have other workspaces defined in Developer Studio.

2. If this is your first time using Developer Studio:
   a. From the main menu, select Help → Help Contents. The Help - Progress Developer Studio window appears.
   b. Open the Progress Developer Studio for OpenEdge Guide and begin with the “Getting Started” and “OpenEdge Projects” topics for basic information about using Developer Studio.
   c. Continue with the “OpenEdge Mobile” topic for information on developing OpenEdge Mobile applications.

The present manual also refers to the online help for more detailed instructions on performing various tasks.

3. If you plan to use the Progress OpenEdge Mobile App Builder to build Mobile Apps, follow the tasks in the online help for configuring the workspace for Mobile. These include (but might not be limited to):
   - Setting user credentials — You need to enter your Progress ID user credentials in Developer Studio the first time you use the Mobile App Builder, and again in the Progress ID login page each time Mobile App Builder starts up without a Progress ID login active in the Web browser.
   - Configuring an external Web browser — This browser runs the Mobile App Builder (after you log in using your Progress ID account) and must support HTML5.

Note: Developer Studio does not support running the Mobile App Builder in an internal browser.
For more information on getting started with the Mobile App Builder, see Chapter 4, "Creating Mobile Apps using JSDoS."

Configuring and managing OpenEdge servers from Developer Studio

OpenEdge Mobile relies on OpenEdge servers to create and work with Mobile services. Before you first start Developer Studio, you might start by configuring these servers, including any database servers that you need for development. This description assumes that you use OpenEdge Management or OpenEdge Explorer to maintain your OpenEdge servers. For more information on these OpenEdge tools, see OpenEdge Management and OpenEdge Explorer: Getting Started. When properly configured, Developer Studio allows you to access these from various views and menus.

Before configuring and managing OpenEdge servers from Developer Studio, you need to configure a connection to OpenEdge Explorer or OpenEdge Management.

To configure an OpenEdge Explorer or OpenEdge Management connection:

1. From the main menu, select Window→Preferences. The Preferences window appears.

2. Select the Progress Explorer→Server→OpenEdge Explorer Connections node. The OpenEdge Explorer Connections page appears.

3. Select the default connection and choose Edit. The Edit Connection Profile window appears.

4. Modify the configuration settings that you require to access your installation of OpenEdge Explorer or OpenEdge Management. Verify, especially, the user name and password.

5. Choose Test Connection to verify that the connection works, and if necessary, re-configure the connection (or your OpenEdge Explorer) settings until it does so.

6. Once your connection works, choose Create Servers and Finish. This ensures that Developer Studio is connected to OpenEdge Explorer or OpenEdge Management and all the default OpenEdge servers, and any others that you have created. This closes the Edit Connection Profile window.

7. In the OpenEdge Explorer Connections page, choose OK to close the Preferences window.

Once you have a working connection established, you can quickly open OpenEdge Explorer or OpenEdge Management from either the Servers or the Progress OpenEdge Server Monitor view, or from the OpenEdge→Admin menu. Using OpenEdge Explorer or OpenEdge Management, you might want to create and configure any OpenEdge servers that you need for OpenEdge Mobile.
OpenEdge installs with default instances for some servers that are appropriate for Mobile development:

- **OE Web Server** — An Apache Tomcat Web server configured with a REST Management Agent to deploy and monitor Mobile Web applications. The default instance is `restmgr1`.

- **AppServer** — An AppServer configured with the state-free operating mode, the only operating mode supported for OpenEdge Mobile. The default instance is `restbroker1`.

- **Database** — You can create and startup any server or servers for OpenEdge-supported databases you need to access from the AppServer and your Mobile projects. Although you can configure OpenEdge RDBMS connections for access from Developer Studio, creating a server for the database allows you to connect and manage the database in multi-user mode from multiple clients, including OpenEdge tools and utilities. There is no default installed database server.

**Note:** You can define database connections for an AppServer either 1) in Developer Studio by choosing from the list of available database connections when you configure AppServer launch configuration settings (typical for Mobile services development), or 2) in OpenEdge Explorer by setting the server startup parameters in the Agent tab during AppServer configuration (typical for Mobile services deployment).

Once you have defined the servers you want to use, you can start and manage the AppServer and OE Web Server instances listed in the **Servers** view of Developer Studio by opening the context menu on each instance.
Creating an OpenEdge Mobile project

You can create a new Mobile project or you can convert an existing OpenEdge project into a Mobile project by performing a Mobile function in the project.

To create a new OpenEdge Mobile project:

1. From the main menu, select **File → New → OpenEdge Project**. The **New OpenEdge Project** wizard appears. This wizard begins by asking you to enter a name for the project (for example, **SportsMobile**) and to select its type from a list, which is **Mobile**. It assigns a default folder for the project based on the location of the workspace.

   **Note:** Note that whatever name you select for the project becomes the basis for the default names of some project components that you create. For example, if you choose to create a Mobile App for the SportsMobile project, the initial default name for the App is SportsMobileApp. Default names for Mobile services are created in a similar way.

2. If you select **Express setup**, and complete the listed prerequisites by clicking any links provided, you can then click **Next** to have Developer Studio automatically build, deploy, and launch a complete Mobile application based on a selected database table. This Express Mobile application runs in your configured HTML 5 browser at the conclusion of the project setup. For more information, see the Developer Studio online help.

3. Otherwise, you can then click **Next** and proceed similarly to create the project through several pages of the wizard, which are described in the “Creating an OpenEdge Mobile project” topic of the online help.

4. When the **Define AppServer content module** page appears, the available AppServers are displayed in a **Supported servers** list. Select a state-free AppServer in the list. The installed default state-free AppServer is **restbroker1**. If you want Mobile services to be updated on the Web server immediately after changes are saved, select **Publish changes immediately**.

5. When the **Create a Mobile service** page appears, select **Create a Mobile service** only if you want a default Mobile service created for the project. You still must go back and add Mobile resources to it after you build them. Otherwise, you can use the **New Mobile Service** wizard to both create the service and add Mobile resources to it after you have built the resources. For information on building Mobile resources, see the “Creating Mobile resources” section on page 66. For information on using the New Mobile Service wizard, see the “Creating Mobile services” section on page 68. If you choose to create the default service, note that the service relative URI is the immediate parent URI for the URIs of all Mobile resources that you add to the service. In the **OE Web Servers** list, select all instances of OE Web Servers where you want the service to be published for testing. The installed default is **restmgr1**.
6. When the Create a Mobile App page appears, select Create a Mobile App only if you want a default Mobile App created with the new Mobile project. The process for doing this is similar to creating a Mobile App using the New Mobile App dialog after the Mobile project has been created. For more information, see the sections on creating a Mobile App in Developer Studio in Chapter 4, “Creating Mobile Apps using JSDOs.”

7. When the Define PROPATH page appears, you can configure the PROPATH similar to setting the PROPATH as part of the project properties.

8. When the Select database connections page appears, you can configure and select existing database connections or add and select new database connections against which the Mobile project allows you to compile ABL source code and from which the project allows you to choose to define given AppServer run configurations.

9. When the Static Web Project page appears, the Context root allows you to specify the default name of the Mobile Web application root, which is also the name for the relative URI of the Web application and the parent URI of all Mobile services that you create in the Web application; and it is also the name of the top-level folder where the Mobile Web application is deployed on the Web server. The Web content folder name allows you to specify the folder for OpenEdge JavaScript files that are deployed with the Web application for a Mobile Web App.

10. At this point, choosing Finish creates the folders and all supporting files for the specified Mobile project. If you chose to create a Mobile App and are using the Mobile App Builder (see Step 6), the Mobile App is also created as a project in Mobile App Builder, which automatically displays the project page in the external browser.
Creating and testing a Mobile service

You create a Mobile service by specifying one or more existing Mobile resources for it to contain. Therefore, you need to create Mobile resources before you can create a Mobile service to contain them. The following general procedure describes how to create and deploy a Mobile service for testing in a development environment.

To create and test a Mobile service:

1. Create one or more Mobile resources from Mobile interfaces that you define from new Business entity classes or existing classes and external procedures that can be instantiated as singleton objects. Classes are inherently designed to be instantiated as singletons. However, procedures need to be coded properly for use as singleton objects. In addition, the ABL routines must meet the requirements of any Mobile operations that they implement.

   For more information, see the “Coding AppServer services for OpenEdge Mobile” section on page 58 and the “Creating Mobile resources” section on page 66.

2. Define the Mobile service with one or more existing Mobile resources. For more information, see the “Creating Mobile services” section on page 68.

3. Publish one or more of the Mobile services to an OE Web Server for testing. For more information, see “Publishing Mobile services for testing” section on page 69.

4. Either write a Mobile App to access the published Mobile services and resources using JSDOs or use a REST client, such as Postman, to access the individual operations of a Mobile resource as REST resources. For more information, see the “URIs for accessing Mobile Web applications, services, and resources” section on page 70.
Coding AppServer services for OpenEdge Mobile

Several options and requirements apply especially to ABL written for an AppServer that implements the services and resources of a Mobile Web application:

- Only a singleton class or procedure object can implement the interface to a Mobile resource.
- The ABL routines that implement Mobile operations have specific coding requirements.

Singleton classes and procedures as Mobile resources

The ABL to implement the interface to a single Mobile resource must be coded in a single class or external procedure that can be executed as a singleton object. A singleton object is a class or external procedure that once initially instantiated, the same instance is shared by all consumers of the class or procedure no matter how many times it is instantiated in a given AppServer session.

When on behalf of a Mobile App, a Mobile Web application executes any Mobile operation of a Mobile resource on an AppServer agent, if the ABL class or external procedure has not yet been instantiated as a singleton object, the agent instantiates it and executes ABL routine that implements the operation. When the operation completes, the object remains instantiated for access by other client requests. So, when the Mobile Web application executes another operation of the same Mobile resource, on the same AppServer agent, the same singleton object is then used to execute the ABL routine for that operation, and so on. If another AppServer agent executes the operation for the same Mobile resource, the same process repeats itself, instantiating the singleton if it does not exist and remaining instantiated for all additional calls to the same resource on that or another AppServer agent where the same object is instantiated. Once all running AppServer agents have executed an operation for that same Mobile resource, they all maintain their singleton objects as long as they continue to run. Again, the process repeats for any additional agent that runs on the AppServer and responds to a Mobile operation request.

ABL classes are inherently coded to be instantiated as singletons. However, external procedures must meet a basic requirement to be instantiated as singletons, and that is they cannot contain any `DEFINE PARAMETER` statements in the main block. They can contain internal procedures and user-defined functions with their own parameters, each of which can implement a Mobile operation exactly like a method of a class.

Note that the singleton coding requirement for external procedures applies only to an external procedure that implements the interface for a Mobile resource and its operations. Any additional procedures or classes that a singleton class or procedure accesses can be implemented as any type of class or procedure that runs on the AppServer.

For more information on singleton procedures instantiated in the context of an ABL client, see the `RUN` statement in *OpenEdge Development: ABL Reference*. Although from the ABL client viewpoint, the description of singleton procedure behavior and coding requirements apply to OpenEdge Mobile as well.
Coding ABL routines to implement a Mobile resource

When you create a Mobile resource in Developer Studio (see the "Creating Mobile resources" section on page 66), you can either create a new class to implement the resource using the New Business Entity wizard or you can use the Define Service Interface wizard to create the resource from an existing ABL class or external procedure coded to run as a singleton. Using these wizards, you can define ABL routines that provide the interface to one or more Mobile resources, each of which supports one or more Mobile operations.

The two basic options that you must define for any class or procedure that implements a Mobile resource include the data model and the Mobile operations that the class or procedure supports. As described previously (see Chapter 1, “OpenEdge Mobile Overview”), a Mobile resource can support a single data model that is managed by its built-in Mobile operations, create, read, update, and delete (CRUD). This data model can be the ABL definition for a single temp-table or a single ProDataSet that contains one or more temp-tables, including any data-relations defined for them. In the Developer Studio wizards, you select the data model from any temp-tables and ProDataSets that are defined in the class or procedure file, or in an external file that you specify. Once completed, the wizard, if directed, inserts the selected data definition in the main block of the class or procedure.

Creating an ABL class with the New Business Entity wizard

When you create a Business entity class to define the Mobile resource (using the New Business Entity wizard), it defines either a single Mobile read operation or a full set of Mobile CRUD operations for the selected data model. If the selected data model is a ProDataSet with multiple temp-tables, it also defines a separate invoke operation for each temp-table that is intended to read the table and return its data to a Mobile App as a JavaScript object.

Each Mobile CRUD operation is defined by a single VOID ABL method whose name consists of a concatenation of the operation name with the name of the resource data model instance, has an empty code block, and has a parameter list that is tailored to reference the selected data model. The exact parameter list for each method is prescribed by the Mobile operation that it implements. For example, if the data for the resource is a ProDataSet named dsCustomer and the read operation is being defined, the method name is created as ReaddsCustomer. You must then add the code for each Mobile CRUD operation to implement each operation according to OpenEdge Mobile functional requirements. For more information, see the “Coding the ABL for Mobile operations” section on page 60.

In addition, when the wizard is directed to generate a Mobile interface, both the first ABL statement in the file and each METHOD statement in the class are preceded by a set of annotations that start with the '@' character and specify how the class and each method is to be accessed and executed as a Mobile resource and operation.

**Caution:** Do not modify the annotations in the ABL source file for a Mobile resource. Doing so can make the resource inaccessible from a Mobile App or otherwise fail to function.
Using existing ABL code with the Define Service Interface wizard

When you define a Mobile resource from an existing ABL class or procedure (using the Define Service Interface wizard), you select the existing class or procedure file, the data model for the new Mobile resource to support, and the existing ABL routines you want to implement Mobile operations for the resource. In this case, you select an operation, and choose the ABL routine you want to implement it. Each ABL routine you have selected for the resource can implement only one operation, whether it is a built-in Mobile CRUD operation or a non-built-in invoke operation. Once you have chosen a routine for every CRUD operation, any remaining routines in the list can each implement an invoke operation. (If you choose all the ABL routines to implement an invoke operation, the Mobile resource then supports no CRUD operations.)

When the wizard completes, it annotates the file and the ABL routines chosen to implement Mobile operations similar to the annotations in the class file and methods annotated for a new Business entity class.

Note that the Define Service Interface wizard does not verify that the existing ABL routines you choose are coded properly to implement a given operation. For a Mobile CRUD operation, the wizard does not even verify that the prescribed parameter list for the operation is correct. If a Mobile CRUD operation has an incorrect parameter list, the operation will not work. So, you might have to revise any existing ABL routines that you chose to implement Mobile CRUD operations to perform the work of that operation, at least according to OpenEdge Mobile functional requirements (described in the following section).

Coding the ABL for Mobile operations

No matter how you obtain the ABL class or procedure to implement a Mobile resource, any ABL routines that you define in the source file to implement Mobile operations must conform to specific coding requirements. Otherwise, a Mobile App cannot access the data using JSDOs. These requirements depend on the operation and the data that you want the resource to provide:

- The ABL routines that implement Mobile CRUD operations must all operate on the same data model.
- Each ABL routine that implements a built-in Mobile create, read, update, or delete operation must have a prescribed parameter list, based on the data model that the Mobile resource supports. When you create a new Business entity for a data model, the wizard generates correct parameter lists for interacting with that data model.
- For the fields of temp-tables in the data model, and for the parameter lists and return types of ABL routines that implement Mobile invoke operations, you can include all supported ABL data types except RAW and RECID. The ABL RAW and RECID data types are not supported by OpenEdge Mobile.
- For all built-in Mobile CRUD operations, the return type for class methods must be VOID and any return values from internal procedures and user-defined functions are ignored.
Table 2 shows a complete description of the prescribed parameter list for each built-in Mobile CRUD operation. The notes in the table describe important additional requirements and limitations that you need to consider.

<table>
<thead>
<tr>
<th>Built-in operation</th>
<th>Prescribed ABL parameter list¹</th>
</tr>
</thead>
<tbody>
<tr>
<td>create</td>
<td>INPUT-OUTPUT { TABLE table-name</td>
</tr>
<tr>
<td></td>
<td>TABLE-HANDLE table-hdl</td>
</tr>
<tr>
<td>read</td>
<td>INPUT filter³ AS CHARACTER ,</td>
</tr>
<tr>
<td></td>
<td>OUTPUT { TABLE table-name</td>
</tr>
<tr>
<td></td>
<td>TABLE-HANDLE table-hdl</td>
</tr>
<tr>
<td>update</td>
<td>INPUT-OUTPUT { TABLE table-name</td>
</tr>
<tr>
<td></td>
<td>TABLE-HANDLE table-hdl</td>
</tr>
<tr>
<td>delete</td>
<td>INPUT-OUTPUT { TABLE table-name</td>
</tr>
<tr>
<td></td>
<td>TABLE-HANDLE table-hdl</td>
</tr>
</tbody>
</table>

1. If the implementing ABL routine is a class method, its return type must be VOID. The return value of any ABL routine is ignored.

2. Because all the built-in operations of a Mobile resource must support the same schema, their implementations all must share either a TABLE, TABLE-HANDLE, DATASET, or DATASET-HANDLE parameter with exactly the same temp-table or ProDataSet schema. **NOTE:** The schema for any ProDataSet (DATASET or DATASET-HANDLE) parameter can have no data-relations defined with the NESTED option.

3. The filter parameter is passed as a query string that is intended to filter the records returned for the temp-table or ProDataSet parameter of a read operation. Your ABL routine can use this value for any purpose, or ignore the parameter entirely. Note that to allow the prescribed mapping to work between the ABL routine for the built-in read operation and the JavaScript method that calls it, you must name this parameter filter in the signature of the ABL routine.
As noted previously, when you define a Mobile resource by creating a new Business entity class, it creates the class methods to implement the Mobile CRUD operations using the correct parameter list, but leaves the code block for each method empty for you to complete. For this purpose, and for any revisions you might need to make to an existing class or procedure you are using to define a Mobile resource, you need to account for certain features of the OpenEdge JSDO:

- A JSDO has an internal data store (JSDO local storage) that is structured to match the data model selected for the Mobile resource that it accesses. So, if the data model is a ProDataSet containing ten temp-tables with data-relations, JSDO local storage is structured to map the data for these ten temp-tables and their data-relations.

- JSDOs currently provide no before-image support, and therefore can only update a single record at a time for a Mobile create, update or delete operation. Therefore, the ABL routines that you use to implement these operations can only update a single input record per invocation on the AppServer.

- The built-in Mobile CRUD operations interact directly with the JSDO local storage. The read operation reads a set of records from its temp-table, or from the temp-tables of its ProDataSet, on the AppServer and loads them into JSDO local storage. The create, update, and delete operations each send only a single record from JSDO internal storage to the AppServer, where they, respectively, add the new record, update the existing record, or delete the existing record from the data source. These operations execute across the network multiple times in order to update multiple records on the AppServer. So, if the data model is a ProDataSet with multiple tables, the ABL routine that implements the operation must query each table for which the operation applies in order to find that one record to create in, update in, or delete from the data source.

- When a Mobile create, update, or delete operation completes, it can only return a single record to JSDO local storage. For example, if an update operation fails, it might return the record with the field values that currently exist in the data source, along with raising an ABL error explaining how the update failed.
Following is a simple ABL class defined for a Mobile resource (with all annotations removed). It implements the built-in Mobile CRUD operations for a Mobile resource defined for a ProDataSet named dsCustomer with the following public methods:

- **Create** — CreatedsCustomer( )
- **Read** — ReaddsCustomer( )
- **Update** — UpdatedsCustomer( )
- **Delete** — DeletedsCustomer( )

In the case, the ProDataSet contains a single temp-table, eCustomer, with a before-table, beCustomer, that is defined for the Customer table in the sports2000 database. The listing of the data model follows the class. All the public methods dispatch their function to private methods that manage the business logic, such as for the filter string passed to the ReaddsCustomer( ) method. The create, update, and delete operations rely on a commitCustomers( ) method to apply the row state from the operation method to the changed record and execute the SAVE-ROW-CHANGES( ) on the corresponding before-table buffer accordingly.

**Note:** For the Mobile create, update, and delete operations, the input side of the INPUT-OUTPUT DATASET parameter replaces the dsCustomer data left over from any prior operation.

---

**Simple Business entity class for a Mobile resource**

```abl
USING Progress.Lang.*.
ROUTINE-LEVEL ON ERROR UNDO, THROW.
CLASS dsCustomer:
    {"dsCustomer.i"}
    DEFINE DATA-SOURCE srcCustomer FOR Customer.
    METHOD PUBLIC VOID ReaddsCustomer(
        INPUT filter AS CHARACTER,
        OUTPUT DATASET dsCustomer):
        THIS-OBJECT:applyFillMethod (INPUT filter).
    END METHOD.
    METHOD PUBLIC VOID CreatedsCustomer(INPUT-OUTPUT DATASET dsCustomer):
        THIS-OBJECT:commitCustomers(INPUT "", ROW-CREATED).
    END METHOD.
    METHOD PUBLIC VOID UpdatedsCustomer(INPUT-OUTPUT DATASET dsCustomer):
        THIS-OBJECT:commitCustomers(INPUT "", ROW-MODIFIED).
    END METHOD.
    METHOD PUBLIC VOID DeletedsCustomer(INPUT-OUTPUT DATASET dsCustomer):
        THIS-OBJECT:commitCustomers(INPUT "", ROW-DELETED).
    END METHOD.
```
Chapter 3: Creating Mobile Services

Note: Although the JSDO on the client currently has no before-image support, the commitCustomers( ) method is coded to be “before-table ready” when JSDOs in a future release do have before-image support. As such, the method uses a FOR EACH statement to mark the row state of each temp-table record (currently only one) sent from the client, and uses another FOR EACH statement to call SAVE-ROW-CHANGES( ) on each record of the AppServer before-image to create, update, or delete the record according to its row state.
Following is the `dsCustomer.i` include file that provides the data model for the simple Business entity class.

**dsCustomer.i for the simple Business entity class**

```plaintext
DEFINE TEMP-TABLE eCustomer NO-UNDO BEFORE-TABLE beCustomer
   FIELD CustNum AS INTEGER
   FIELD Name AS CHARACTER FORMAT "X(20)"
   FIELD Address AS CHARACTER
   FIELD Phone AS CHARACTER
   FIELD SalesRep AS CHARACTER
   FIELD CreditLimit AS DECIMAL
   FIELD Balance AS DECIMAL
   FIELD State AS CHAR
   FIELD numOrders AS INT
   INDEX CustNum IS UNIQUE PRIMARY CustNum
   INDEX Name NAME.

DEFINE DATASET dsCustomer FOR eCustomer.
```
Creating Mobile resources

As noted in the “Coding AppServer services for OpenEdge Mobile” section on page 58, you can create a Mobile resource from Developer Studio in the following ways:

- Create a new Business entity class for the resource using the New Business Entity wizard.
- Define a mobile interface for the resource in an existing ABL class or procedure using the Define Service Interface wizard.

The following sections describe how to do this, with some additional notes on the available options. Also, see the tasks for defining Mobile resources in the Developer Studio online help.

Using the New Business Entity wizard

The New Business Entity wizard creates a Mobile resource from a new Business entity class that implements specified Mobile operations with empty methods defined with the signatures required for the corresponding operations.

To create a Mobile resource using the New Business Entity wizard:

1. From the Progress Explorer view, right-click an OpenEdge Mobile project and select New → Business Entity on the context menu. The Create a Business entity class page of the New Business Entity wizard appears.

2. Enter the Business entity name, select your choice of class options and click Next through the wizard to define the Business entity.

3. On the Select a schema file page enter the resource name (or accept the default), select the access methods (Mobile operations to define) and the schema (data model), whether to expose the Business entity as a Mobile resource, and a resource URI (or accept the default). If you select Read-only operations for the access methods, only the built-in Mobile read operation is defined for the selected data model. However, selecting CRUD operations defines all of the Mobile create, read, update, and delete operations for the selected data model. Note that for a ProDataSet with multiple tables, the wizard always defines a separate Mobile invoke operation to read data from each temp-table in the ProDataSet. If you select Expose as mobile service (the default) the Business entity is created with the annotations to define it as a Mobile resource. Otherwise, it is created without these annotations and is not a Mobile resource; to make it a Mobile resource, you then need to use the Define Service Interface wizard to make the existing Business entity class a Mobile resource (see the following section), where you can also define additional Mobile invoke operations.

4. Clicking Finish generates the Business entity class with empty methods for the specified operations.

You need to add the necessary code to the class to implement the Mobile operations. For more information, see the “Coding AppServer services for OpenEdge Mobile” section on page 58.
Using the Define Service Interface wizard

The Define Service Interface wizard creates a Mobile resource from an existing class or procedure by selecting existing ABL routines (class methods, internal procedures, or user-defined functions) to implement selected Mobile operations.

To create a Mobile resource using the Define Service Interface wizard:

1. From the Progress Explorer view, right-click an OpenEdge Mobile project and select Progress OpenEdge → Define Service Interface on the context menu. The Define Service Interface wizard appears.

2. From the Definition Mode drop-down, select Mobile.

3. From the Workspace resources list, select the class or procedure files you would like to define as Mobile resources. For each file, select the ABL routines that you want to use to implement Mobile operations. Click Next.

4. From the Edit Annotation page, you can select a file, then specify the annotations to define both the file as a Mobile resource and selected methods to implement its Mobile operations. In the Main Annotation tab, you can enter a resource name and URI (or accept the defaults), select a schema (data model) for the resource. In the CRUD annotations and Invoke annotations tabs, you can choose which among the selected ABL routines in the file implements a given Mobile operation. A given routine can implement only one operation. Once chosen for a CRUD or invoke operation, it can not be chosen again for another CRUD or invoke operation.

5. Once you have completed Step 4 for all the class and procedure files you have selected to define as Mobile resources, clicking Finish annotates all the files as Mobile resources.

However, note that this wizard does not check that the ABL routines you have annotated have the correct parameter lists defined and otherwise implement the functionality of any of the built-in Mobile CRUD operations according to OpenEdge Mobile functional requirements. So, you might have to revise the implementing ABL routines accordingly. For more information, see the "Coding AppServer services for OpenEdge Mobile" section on page 58.
Chapter 3: Creating Mobile Services

Creating Mobile services

Once you have created Mobile resources, you can create Mobile services that provide these resources and which you can publish for testing or deploy to an external Apache Tomcat Web server.

Using the New Mobile Service wizard

The New Mobile Service wizard allows you to define a new Mobile service with selected Mobile resources.

Note: If you created a default Mobile service when you created a Mobile project, you can also modify that service to add its Mobile resources. For more information, see the “Modifying and deleting Mobile services” section on page 68.

To define a new Mobile service:

1. From the Progress Explorer view, right-click an OpenEdge Mobile project and select New → Mobile Service on the context menu. The New Mobile Service wizard appears. Click Next from page to page.

2. On the first page, you must specify a name for the service and a relative URI, which defaults to the service name with a preceding slash (/). You can also change the project into which to generate the service artifacts. Select at least one OE Web Server instance where you want to publish the service for testing.

3. On the next page, you can select any of the available Mobile resources to include in the service.

4. Clicking finish generates all the artifacts for the service, including its JSDO catalog file, which a Mobile App needs to load to access the service.

Once you create a Mobile service, you can modify, delete, or publish the service for testing or deployment. For more information publishing the service, see the “Publishing Mobile services for testing” section on page 69.

Modifying and deleting Mobile services

You can edit or delete a Mobile service by right-clicking on the service and choosing Delete or Edit from the context menu. If you choose Edit, the Edit Mobile Service wizard appears, which works much like the New Mobile Service wizard to modify some of the options to create the service.
Publishing Mobile services for testing

When you create a Mobile project in Developer Studio, if you choose the option to publish changes immediately, as long as the OE Web Server that you have selected for a Mobile service is running, when you create or in any way update the Mobile service and its resources, the Web server publishes the service.

If the OE Web Server is not running when you create the Mobile service, or you did not select the project option to publish changes immediately, you can use the Servers view to publish the service by first starting the AppServer for the project (right-click on the AppServer and select Start). Once the AppServer is started and synchronized, you can start the OE Web Server instance for the Mobile service (right-click on the OE Web Server and select Start). Once the OE Web Server is started and synchronized, right-click on it and select Publish, and any Mobile services not yet published should publish. Developer Studio publishes each Mobile service in the project as a separate WAR file, with the name of the Mobile service as its filename, to the selected OE Web Server for testing.

To test a published Mobile service, you either need to write a Mobile App or use a REST client, such as the Postman REST Client, and use the absolute URIs of Mobile operations to send HTTP requests to execute those operations. For more information, see the “URIs for accessing Mobile Web applications, services, and resources” section on page 70.

For information on deploying Mobile services for production, see Chapter 6, "Deploying Mobile Applications.”
URIs for accessing Mobile Web applications, services, and resources

The URIs required to access Mobile Web applications, services, and resources depend on what type of client you use to access them, a Mobile App using JSDOs or another REST client, such as you might use for testing and debugging Mobile resources. The following sections describe the requirements for forming these URIs:

- Mobile URIs for Mobile App access
- Mobile URIs for testing access from REST clients
- Using a REST client to invoke a Mobile operation

Mobile URIs for Mobile App access

For a Mobile App to access the Mobile services and resources of a Mobile Web application, it needs only the relative or absolute URI of both the Mobile Web application and the JSDO catalogs for any Mobile services that the Mobile App accesses. After logging into the Web application and loading the required JSDO catalogs, the Mobile App has only to call methods on JSDOs created for each Mobile resource to access operations of the resource. No other URIs are required.

The following syntax describes the URIs that you need to log into a Mobile Web application and access its services using a JSDO:

Syntax

```
scheme://host[:port]/web-app[\static-resource]
```

The beginning of every absolute URI:

- **scheme** — A URI scheme supported for OpenEdge Mobile, which can be either HTTP or HTTPS.
- **host** — The host name or IP address, which is typically localhost for testing Mobile services on the same machine as both the Web browser and the OE Web Server that hosts the Mobile services.
- **port** — Optional port where the Web server is listening for HTTP requests. The default value, if you do not specify this option, is 80. The default value configured for the OE Web Server is 8980.

The host[:port] can also be replaced by a Web domain (such as, www.progress.com) that accesses the required host and port.
The relative URI for the Mobile Web application, where `web-app` is the name of the Mobile Web application that contains your Mobile service or services, and serves as the root URI for all Mobile resources provided by the Mobile Web application. By default, this is the filename of the WAR file that you use to publish or deploy the Mobile services to your OE Web Server, and it is also the name of the Web server folder in which all of the Web application's Mobile services and Web resources are deployed. As noted during development (see the “Publishing Mobile services for testing” section on page 69), Developer Studio publishes each Mobile service defined for a project in its own WAR file with the filename set to name of the Mobile service. For production deployment, you can export multiple Mobile and REST services in a single WAR file, which by default has the name of the Mobile project. Note that during production deployment, a Web administrator can change the filename of the WAR file and the name of the Web application to a different value.

**Note:** During development, you cannot change the name of the WAR file that Developer Studio publishes for each Mobile service in a project.

If you run the Mobile App in a Web browser and deploy it as a Mobile Web App to the same OE Web Server as the Mobile services, the Mobile App only needs to access this relative URI to log into the Mobile Web application and access Mobile services.

If you install and run the Mobile App as a Mobile Native App in a native device container, or deploy and run it as a Mobile Web App from a different Apache Tomcat Web server from where the Web services are deployed, the Mobile App must access the absolute URI to log into the Mobile Web Application.

The relative URI for a static file or Web page that the Mobile App accesses from the Mobile Web application. For example:

- `/static/mobile/service-name.json` — The default relative URI of the JSDO catalog for a Mobile service, where `service-name` is the name of the service. The Mobile App can load this catalog to access Mobile resources provided by the service after logging into the Mobile Web application. To load the catalog, it can use the relative URI (starting with the Mobile Web application, `/web-app`) or the absolute URI, depending on the same conditions for using a relative or absolute URI for logging into the Mobile Web application.

- `/static/home.html` — The default relative URI of a non-UI login target available to support login to a Mobile Web application using HTTP Basic Authentication.

- `/static/auth/login.html` — The default relative URI of a login form page available to support login to a Mobile Web application using HTTP Forms Authentication.

For more information on logging into a Mobile Web Application from a Mobile App, see Chapter 4, "Creating Mobile Apps using JSDOs."
Mobile URIs for testing access from REST clients

After first developing and publishing Mobile services, you might want to test the Mobile resources they provide before you create a Mobile App to access them using JSDOs. You can do this using various REST clients, such as the Postman REST Client, that allow you to access Mobile operations directly as REST resources. (Remember that a single Mobile resource encapsulates several REST resources as Mobile operations.) In this case, you need to know the exact absolute REST URI for the Mobile resource that represents each Mobile operation.

As described in previous sections of this chapter, when you create Mobile resources and the Mobile services to contain them, you assign a relative URI (or use the default) for each Mobile service and resource that you define. In addition, each operation that you define for a Mobile resource is assigned a prescribed relative URI as part of the resource annotations added to the ABL.

The absolute URI to access a given operation is a concatenation of all these relative URIs with the absolute root URI of the deployed Mobile Web application, as specified using the following syntax:

**Syntax**

```
scheme://host[:port]/web-app/rest/service-name/resource-name[op-element]
```

In this context, as described previously (see the "Mobile URIs for Mobile App access" section on page 70).

```
/web-app
```

The relative URI for the Mobile Web application, where `web-app` is the name of the Mobile Web application that contains your Mobile service or services, and serves as the root URI for all Mobile resources provided by the Mobile Web application. By default, this is the filename of the WAR file that you use to publish or deploy the Mobile services to your OE Web Server, and it is also the name of the Web server folder in which all of the Web application’s Mobile services and Web resources are deployed. As noted during development (see the “Publishing Mobile services for testing” section on page 69), Developer Studio publishes each Mobile service defined for a project in its own WAR file with the filename set to name of the Mobile service. For production deployment, you can export multiple Mobile and REST services in a single WAR file, which by default has the name of the Mobile project. Note that during production deployment, a Web administrator can change the filename of the WAR file and the name of the Web application to a different value.

**Note:** During development, you cannot change the name of the WAR file that Developer Studio publishes for each Mobile service in a project.
URIs for accessing Mobile Web applications, services, and resources

/rest

This is a literal relative URI that identifies the root for all OpenEdge REST resources provided by a Mobile Web application.

**Note:** Unlike the /static relative URI that identifies the physical location of most static Web resources, such as Web pages, this URI does not represent a physical folder on the Web server, but simply identifies a dynamic point in a URI where the relative URI to a given REST resource begins.

/service-name

The relative URI for the Mobile service, where service-name defaults to the name you assign the service when you define it in Developer Studio. You can also assign a different name for the URI at the time the Mobile service is defined. For more information, see the “Creating Mobile services” section on page 68.

/resource-name

The relative URI for the Mobile resource, where resource-name defaults to the name you assign the resource when you define it in Developer Studio. You can also assign a different name for the URI at the time the Mobile resource is defined. For more information, see the “Creating Mobile resources” section on page 66.

/op-element

A relative URI or a URI query parameter that is specified to identify only certain Mobile operations, as follows:

- **?filter=filter-string** — For the read operation, identifies the INPUT filter parameter value passed to the implementing ABL routine, where filter-string is the string value you type, quoted if necessary.

- **/routine-name** — For an invoke operation, where routine-name is the case-sensitive name of the ABL routine that implements the specified invoke operation

Note that all other INPUT (and the input side of INPUT-OUTPUT) parameters passed to the implementing ABL routines for Mobile operations are passed in the body of the HTTP request. For more information, see the following section on using Mobile URIs in a REST client.
Using a REST client to invoke a Mobile operation

Once you have constructed the correct URI for the REST resource that represents a given Mobile operation, you can specify it as part of the HTTP request to invoke the operation from a REST client, such as Postman. However, a REST client requires four basic pieces of information to send an HTTP request:

- **URI** — As described in the previous section
- **Media type** — Always `application/json` for Mobile operations
- **HTTP method** — As specified for each Mobile operation:
  - **Create** — `POST`
  - **Read** — `GET`
  - **Update** — `PUT` (with no `op-element` in the URI—see previous section)
  - **Delete** — `DELETE`
  - **Invoke** — `PUT` (for all invoke operations, each of which is identified and distinguished from the others by the name of the implementing ABL routine specified as the `op-element` in the URI—see previous section)
- **Other HTTP request components** — Especially other `INPUT` parameters to include in the body of the HTTP request

Note that, other than the `filter` parameter that is passed as the `op-element` in the URI of a Mobile read operation request (see previous section), all other `INPUT` (and the input side of `INPUT-OUTPUT`) parameters passed to the implementing ABL routines for Mobile operations are passed in the body of the HTTP request. This includes the JSON object for an `INPUT-OUTPUT` temp-table or ProDataSet parameter that is passed for the Mobile create, update, and delete operations, and a JSON object with properties for passing the values of any `INPUT` and `INPUT-OUTPUT` parameters for Mobile invoke operations.

For the REST client, the input JSON objects specified for relational data must conform to the structure defined by OpenEdge for passing the JSON representations of temp-tables and ProDataSets, as appropriate. For more information, see *OpenEdge Development: Working with JSON*. Note that this structure is similar to the structure used to return the data for a Mobile read operation (`GET`). For an invoke operation, the property names in the simple JSON object must have the same case-sensitive names as the corresponding parameters defined in the implementing ABL routines.

Note also that for the Mobile create, update, and delete operations, all of which have the same type of input JSON object for relational data, only the HTTP method specified for the HTTP request distinguishes one Mobile operation from the other.
As described in Chapter 1, “OpenEdge Mobile Overview,” OpenEdge Mobile supports the development of Mobile Apps with access to OpenEdge data resources on an AppServer using OpenEdge JavaScript data objects (JSDOs), which are instances of the OpenEdge JavaScript class, `progress.data.JSDO`. With the help of additional OpenEdge JavaScript classes and objects, JSDOs access these data resources through Mobile services running on a Web server and make that data available through the Mobile App. For a reference to the basic OpenEdge JavaScript classes and objects available for access in Mobile Apps, see Appendix B, “OpenEdge JavaScript Class and Object Reference” and Appendix C, “OpenEdge JavaScript Properties, Methods, and Events Reference.” The Progress OpenEdge Mobile App Builder, which provides advanced support for building OpenEdge Mobile Apps, supports additional built-in client services for managing JSDOs.

Using OpenEdge Mobile, you can build two basic types of Mobile Apps that access the same Mobile resources, but which run on different types of client platforms: Mobile Web Apps that run in a Web browser on any client platform or device, and Mobile Native Apps, packaged using hybrid technology, that run in a native device container deployed to either an Apple iOS or Google Android device. You can build these Mobile Apps using the Mobile App Builder, which provides visual services for building Web pages and mapping JSDO data to Web page elements, or you can use basic HTML and JavaScript coding tools, such as those available in Progress Developer Studio for OpenEdge (Developer Studio).
This chapter describes how to use the features of a JSDO and associated OpenEdge Mobile JavaScript objects to access Mobile services, regardless of the type of Mobile App or coding platform. It then describes how to get started with building Mobile Apps using either the Mobile App Builder and its data mapping services or more basic HTML coding tools in the following sections:

- Creating a Mobile App from Developer Studio
- Getting started with the Mobile App Builder
- JSDO overview
- Creating a JSDO for a Mobile resource
- Accessing built-in CRUD operations
- Accessing non-built-in invoke operations
- Managing user login sessions
- Getting started with other HTML coding tools
- Publishing Mobile Apps for testing
Creating a Mobile App from Developer Studio

You can begin to create a Mobile App in Developer Studio when you first create an OpenEdge Mobile project using the New OpenEdge Project wizard, or after the project is created using the New Mobile App dialog. The basic procedure for these tasks is similar either way. This procedure basically creates the folder for the Mobile App HTML and JavaScript files in the Mobile project from which to publish them as a Mobile Web App on an OE Web Server or to export them in a WAR file for deployment to an external Web server. This location is shared with the Progress OpenEdge Mobile App Builder, where it saves the files that you create in the Mobile App Builder project. For information on how to create a Mobile App as part of creating a new OpenEdge Mobile project, see Chapter 3, “Creating Mobile Services.”

To create a new Mobile App for an existing Mobile project:

1. From the Project Explorer view, right-click on an OpenEdge project for which you want to define the Mobile App, then select New → Mobile App on the context menu. The New Mobile App dialog appears.

2. Here, you can:
   - Enter the Mobile App name. Note that any name you choose for a Mobile App must be unique across all projects and workspaces of Developer Studio.
   - Change the Mobile App project in which it is created.
   - Specify a template with which to build the Mobile App. The Mobile App type label only identifies the template used in Mobile App Builder to design the pages of the App for a given general device category (such as a phone or tablet) or for an application example (for example, a Twitter or weather app). It has nothing to do with the type of deployment for the Mobile App (such as a Mobile Web App a Mobile Native App).
   - Select a backup from which to create the Mobile App.
   - Select an OE Web Server instance (or instances) on which to publish the Mobile App. The installed default is restmgr1.

3. If you have not yet entered Progress ID user credentials in Developer Studio, this page displays an error message reminding you to do so, and also provides a link to the Mobile App Builder preferences page to do so. You only need to provide these user credentials if you intend to use the Mobile App Builder to create your Mobile Apps. If you use other HTML and JavaScript coding tools, such as the JSDT in Developer Studio, you do not need to enter any user credentials. For more information on the JSDT, see the JavaScript Development Guide in the Developer Studio online help. If you have not yet configured an external HTML5 Web browser to run the Mobile App Builder or to test Mobile Apps, you can also click a link to the Web Browser preferences page to do that.

4. When you click Finish, Developer Studio creates an entry for the new Mobile App in the Mobile Apps folder of the project. If you are set up to build the App in Mobile App Builder, the configured external browser opens to the Progress ID login page, where you can login to open the Mobile App Builder, where you will find a new project for the Mobile App.
Getting started with the Mobile App Builder

Using the Progress OpenEdge Mobile App Builder you can visually build a single client that can be deployed as a Mobile Web App, an iOS App or an Android App.

The Mobile App Builder is a cloud-based development tool integrated with Progress Developer Studio for OpenEdge. A Developer Studio project can contain one or more Mobile Apps, each of which is associated with a unique Mobile App Builder project. The Mobile App Builder is secure and requires a Progress ID along with an active subscription.

The Mobile App Builder simplifies the development of a Mobile App by handling much of the complexity of jQuery Mobile, HTML5, CSS, JavaScript, and Native App deployment (managed with Apache Cordova). The Mobile App Builder provides page templates, along with jQuery Mobile components, to make it fast and easy to prototype and build the UI of the Mobile App. Simply select any component and drag and drop it into the phone or tablet image. Once a component is added, you can set various properties available on the right-hand side. The Mobile App Builder generates standards-based code for maximum compatibility.

The Mobile App Builder hides the complexity of building the UI as well as adding behavior to the pages such as, event handlers, page and link navigation, and REST service invocation. These can all be done without the need for you to write any JavaScript. In the event that you need to write custom logic, you can always provide your own JavaScript for the page.

One of the most powerful features of the Mobile App Builder is the tight data-binding with OpenEdge ProDataSets and temp-tables. As described in Chapter 3, “Creating Mobile Services,” the data and ABL business logic are accessible by creating an OpenEdge Mobile service. Within the Mobile App Builder, you simply upload the JSDO catalog associated with the Mobile service, select the resources and data model (schema) that you want to use in the Mobile App, and Mobile CRUD operations and custom business logic for the selected resources are automatically imported as JSDO Services. JSDO Services are coding services that you can use to invoke Mobile operations to pass data back and forth between the Mobile App and the OpenEdge AppServer, or to invoke custom business logic. All communication and data transformation is automatically handled by the JSDO Services, with the need for little or no additional JavaScript code on the client. In addition, data returned from a JSDO Service can be more easily mapped to UI components than by working directly with JSDO local storage. For more information on using JSDO Services, see Chapter 5, “Using JSDO Services in the Mobile App Builder.”

Once the Mobile App is built, it can be previewed for immediate feedback and collaboration. The preview is a Mobile Web App that runs in an emulator. This can be automatically deployed on the Mobile App Builder server in the cloud or it can be downloaded locally and deployed to a local Web server of your choice. Other available deployment choices include building an iOS App deployed through the Apple AppStore or building an Android App deployed through Google Play. For more information on these Mobile App deployment options, see Chapter 6, “Deploying Mobile Applications.”
**JSDO overview**

A JSDO is an object designed to simplify access to OpenEdge relational data in a Mobile App. It does this by providing JavaScript methods to execute the Mobile operations supported by a single Mobile resource, and supports an internal data store (*JSDO local storage*) to cache the OpenEdge data defined, and initially returned to the Mobile App, by the Mobile resource.

The JSDO relies on a JSON catalog file (*JSDO catalog*) that defines the Mobile resource it is accessing. This resource definition includes the schema (data model) for the OpenEdge data supported by the resource as well as the definitions for JSDO methods to call the Mobile operations of the resource. The schema of the Mobile resource, therefore, determines both the structure of the data in JSDO local storage and how Mobile operations can interact with it. Other JSDO methods allow the Mobile App to read and manipulate the data in JSDO local storage for ultimate update on the AppServer by calling the supported Mobile operations. In addition, the JSDO’s local storage provides features that facilitate mapping its data to HTML elements of a Mobile App.

The following sections briefly describe:

- **Supporting OpenEdge classes and objects**
- **How a JSDO maps to a Mobile resource**
- **How JSDO local storage works**
- **Methods of a JSDO and the JSRecord object**
- **Asynchronous and synchronous execution**
- **Properties of a JSDO**
- **Requirements for using a JSDO**

**Supporting OpenEdge classes and objects**

OpenEdge Mobile provides the following classes and objects to support JSDO creation and access:

- **`progress.data.JSDO` class** — Allows you to create JSDOs for Mobile resources that can execute the resource-supported Mobile operations that exchange the data defined by these resources between a Mobile App and an AppServer where the data is stored, as described in this “JSDO overview” section and the current chapter.

- **`progress.data.Session` class** — Allows you to create a `Session` object that manages a user login session between the Mobile App and a Mobile Web application. This enables a JSDO to access the resources that a Mobile service supported by the Mobile Web application provides, as described in this “JSDO overview” section. For more information, see the “Managing user login sessions” section on page 106.

- **`JSRecord` object** — References a JavaScript record object from JSDO local storage, as described in this “JSDO overview” section.
• **Request object** — Contains data both sent in HTTP requests for Mobile operations when they are executed and returned in HTTP responses when the Mobile operation execution completes, as described in this “JSDO overview” section. For more information, see the “Accessing built-in CRUD operations” section on page 95 and the “Accessing non-built-in invoke operations” section on page 104.

• **A set of JSDO Services built into the Mobile App Builder** — For mapping JSDO data to any HTML page elements you can create, and supporting the marshalling of that data between the Web page and Mobile operations, with little or no need for additional JavaScript code. For more information, see Chapter 5, “Using JSDO Services in the Mobile App Builder.”

• **progress.ui.UIHelper class** — Provides basic features for mapping JSDO data to Web page elements in Mobile Apps built with HTML tools other than the Mobile App Builder. For more information, see the “Getting started with other HTML coding tools” section on page 111.

### How a JSDO maps to a Mobile resource

A JSDO is an instance of the **progress.data.JSDO** JavaScript class that you create to access exactly one Mobile resource provided by a Mobile service. As described in Chapter 3, “Creating Mobile Services,” a Mobile resource provides access to either a single temp-table or a single ProDataSet with one or more temp-tables on an OpenEdge AppServer. The exact type of data that a Mobile resource provides depends on the data model selected to create the resource. A Mobile resource is also created with a set of built-in operations to access its data model. These can include a complete set of create, read, update, and delete operations (CRUD), or only a single read operation, depending on the options chosen to create the Mobile resource. In any case, the same basic set of Mobile operations access the data for every Mobile resource regardless of the data model it supports. The ABL routines that implement these operations have prescribed signatures that depend on the type of data model selected for the Mobile resource.

The prescribed relationship between the built-in operations and the data model of a Mobile resource allows the **JSDO** class to provide a corresponding set of built-in JavaScript methods that implicitly map to the built-in operations of a Mobile resource, no matter what data model it supports. In addition, for every JSDO, the internal structure of the JSDO local storage reflects the schema of the particular data model supported by the Mobile resource. Therefore, the built-in methods of a JSDO work on the data in its local storage according to the schema defined for the supported data model.

The basic unit of data for a Mobile resource is the temp-table record, which is represented in JSDO local storage as a **JavaScript record object**. The built-in create, update, and delete operations of a Mobile resource operate on one AppServer temp-table record at a time, according to the implementation of the corresponding ABL routines. The read operation returns a set of records from the AppServer for one or more temp-tables, depending on the data model. The set of records that the Mobile read operation returns depends on the implementing ABL routine and an optional filter parameter that you pass to it from the Mobile App. The corresponding built-in JSDO methods call these operations accordingly, loading temp-table records into JSDO local storage from the AppServer, or creating, updating, and deleting records on the AppServer from posted changes to the record objects in JSDO local storage.
A Mobile resource can also support non-built-in invoke operations which can be implemented by any ABL routines defined for the resource on the AppServer except routines that already implement built-in Mobile operations. Routines that implement invoke operations can have any signature defined with supported ABL data types. A JSDO created for a Mobile resource with invoke operations has corresponding non-built-in invocation methods that map to these invoke operations. Calling an invocation method on the JSDO calls the implementing ABL routine, passing the required input parameters and returning its output parameters and any return value to the Mobile App with no direct effect on JSDO local storage. The Mobile App can do whatever it needs to do with the results of an invocation method, including merging them with JSDO local storage, if appropriate.

For more information on coding the implementing ABL routines of Mobile operations on the AppServer, see Chapter 3, “Creating Mobile Services.”

How JSDO local storage works

JSDO local storage stores temp-table records as JavaScript record objects according to the schema of the data model supported by the Mobile resource of the JSDO. If the data model is for a single temp-table, local storage can contain only record objects for the specified temp-table.

If the data model is for a ProDataSet, local storage can contain record objects for the temp-tables defined for the ProDataSet. By default, record objects for a ProDataSet are maintained in local storage according to any data-relations defined for the ProDataSet. This means, for example, that when a JSDO method finds a record object of a parent temp-table, if a method is then called to search through the record objects of a temp-table that is a child of that parent, the search will find only record objects that are related to the record object found in the parent; if new record objects are added to the same child temp-table, they are added with key fields set implicitly in relation to the parent record object. The JSDO also supports the run-time option (by setting the useRelationships property) of toggling between honoring these data-relations and ignoring them when accessing temp-table data in local storage.

Table and field references

You can access the data in local storage using table references. A table reference is a property on the JSDO that references a given JavaScript temp-table object as defined by the schema of the data model. So, local storage contains one table reference property for each temp-table referenced in local storage. The name of each property is the same as the name of a corresponding temp-table defined in the schema selected to define the Mobile resource in Developer Studio (see Chapter 3, “Creating Mobile Services”), and it is specified with the same letter case as the temp-table name in the selected schema.

JSDO methods that operate on local storage operate either on the entire data store, in which case they are called on the JSDO itself, or on one table reference at a time, in which case they are called directly on the corresponding table reference property. For example, given a JSDO referenced by dsOrderEntry whose local storage references several temp-tables of a ProDataSet, including ttCustomer, two JSDO methods might be called as follows:

```javascript
dsOrderEntry.fill();
dsOrderEntry.ttCustomer.foreach( function ( record-object ) { ... } );
```
In this example, the `fill( )` method is called on the `dsOrderEntry` JSDO to load the available data for the Mobile resource into local storage by calling the built-in Mobile read operation. Then, the `foreach( )` method is called on the `ttCustomer` property of the JSDO to loop through all the record objects loaded for the `ttCustomer` temp-table, allowing each one to be accessed from within the function that is passed as a parameter.

**Working records**

When a JSDO method is called, depending on the method and the situation, it might result in setting a working record for one or more of the table references. A working record is a record object for a given temp-table in local storage that is available to reference implicitly using the table reference. If a table reference has a working record set for it, you can then reference any field value in the corresponding record object using a corresponding field reference property on the table reference. The name of the field reference property is the same as the temp-table field name in the schema defined for the Mobile resource. You can also use the working record on a table reference to return the record object for the working record and store it for future reference.

For example, when the `foreach( )` method in the previous example returns from execution (based on the return value of its function parameter), it leaves a working record set for the `ttCustomer` table reference. You can assign a value to the `Name` field of this working record using the `Name` field reference property on the table reference:

```javascript
dsOrderEntry.ttCustomer.Name = "Evellyn Doe";
```

This is one of the supported mechanisms to update a field value in local storage. The other is to call the `assign( )` method directly on a table reference or on a record object, itself.

Also, if local storage has a ProDataSet data model with active data-relations and a working record is set in a parent table reference, a search through the records of a child table reference reflects the data-relation. For example, with a working record set for a `ttCustomer` that is a parent of `ttOrder`, calling the `foreach( )` method on `ttOrder` loops only through records related to the working record in `ttCustomer`.

**Note:** Table references for temp-tables referenced in local storage use a flat reference model. In other words, regardless of data-relations, you reference the property for each table reference directly on the JSDO (`dsOrderEntry.ttOrder.Ordinalnum`).

Depending on the results of JSDO methods that you call, local storage either maintains a working record for each table reference or leaves the working record for one or more table references undefined. Essential programming for a JSDO involves calling appropriate methods to locate working records in local storage in order to read and modify their contents. The documentation for each JSDO method (see Appendix C, "OpenEdge JavaScript Properties, Methods, and Events Reference") indicates whether and how it leaves a working record when it completes execution.
If a method leaves a working record, you can reference the field values of the record object using field reference properties on the table reference, as described in the previous section. You can also use the `record` property on the table reference to return a reference to a `JSRecord` object for the working record that you can reference directly or store separately to access the record object later. A `JSRecord` object provides a `data` property that references a separate JavaScript object with the actual field reference properties for the record. Also as a convenience, if the JSDO supports only a single temp-table, you can access the `record` property for the working record directly on the JSDO reference itself.

So, using the previous example with `ttCustomer`, where it is the only temp-table in local storage, you can access the `Name` field using a `JSRecord` object reference in the following ways:

```javascript
var custName = dsOrderEntry.record.data.Name; // Single temp-table only
var custName = dsOrderEntry.ttCustomer.record.data.Name;

var custRecord = dsOrderEntry.ttCustomer.record; // Stored for later reference
var custName = custRecord.data.Name;
```

Once stored, an existing record object remains available using the stored `JSRecord` reference even when it is no longer the working record for the temp-table. Note that using the `data` property to write a value to a field is not recommended, because the record object is not marked as changed in local storage and won’t be updated on the AppServer. To update a field value on a `JSRecord` object reference so the change is made on the AppServer, call the `assign( )` method directly on the `JSRecord` reference.

**Note:** One reason to use the `record.data` property on a table reference to read a field value is when the field has the same name as a JSDO method that you can call on a table reference.

**Caution:** Because the record object is not marked as changed in local storage, never use the `data` property on a `JSRecord` reference to update the value of a field. The change will never be posted to AppServer. Use the `data` property to read the field value only.

**Record IDs**

One difference between the JavaScript record objects in JSDO local storage and the temp-table records that they represent is that OpenEdge creates each JSDO record object with a local record ID. This is an internal field reference with the OpenEdge-reserved name `_id` that uniquely identifies the record in JSDO local storage. This record ID has no relationship to the internal `RECID` and `ROWID` values maintained for the records of an OpenEdge database. Instead, this record ID is used by the client JSDO Services built into the Mobile App Builder, and also by the `progress.ui.UIHelper` class, to map the record objects in JSDO local storage to the HTML elements of Mobile Apps. The JSDO provides the `getId( )` method for you to retrieve this record ID from any record object in local storage and store it for future reference in order to efficiently retrieve that record object again from local storage.

**Caution:** Do not change the value referenced by `_id`. Otherwise, any Mobile App UI managed by OpenEdge can have unpredictable behavior.
Methods of a JSDO and the JSRecord object

Every JSDO has a number of methods, many of which interact with JSDO local storage. Some are called on the JSDO itself, while most are called on a JSDO table reference. Most execute synchronously, but some execute asynchronously as noted. For more information, see the "Asynchronous and synchronous execution" section on page 87. A few JSDO methods are also available on a JSRecord object.

A JSDO includes methods for:

- Executing built-in Mobile operations and calling their implementing AppServer routines:
  - `fill()`: Executes the read operation of a Mobile resource, loading the temp-table records sent from the AppServer into JSDO local storage according to the resource data model; the first record of every loaded temp-table reference is set as its working record.
  - `saveChanges()`: Executes the create, update, or delete operation of a Mobile resource for each record that has changed in JSDO local storage since the last execution of the `fill()` or `saveChanges()` method; no working records for any table references in local storage are set.

These methods are always called on the JSDO and execute asynchronously.

- Executing non-built-in Mobile invoke operations and calling their implementing AppServer routines. There is a uniquely named JSDO method defined for each invoke operation in the Mobile resource; these methods have no effect on JSDO local storage.

These methods are always called on the JSDO and, by default, execute asynchronously, but can optionally execute synchronously. To pass input parameters to the implementing ABL routine, you pass an input object to the invocation method that contains properties with the values for any ABL input parameters. A property for an ABL input parameter has the same name (and letter case) as the corresponding input parameter, and has a JavaScript data type that maps to the ABL data type of the input parameter (see Appendix A, "ABL to JavaScript Data Type Mapping")

- Updating JSDO local storage:
  - `add()`: Called on a JSDO table reference, this method creates a new record object for the temp-table in JSDO local storage; the new record is set as the working record for the table reference.
  - `addRecords()`: Called either on a JSDO or on a JSDO table reference, this method merges record objects from a merge object passed as a method parameter with the specified existing record objects in JSDO local storage. The merge object must follow the same schema as JSDO local storage itself. Merge modes determine how to handle record objects with duplicate key fields, if specified. The case sensitivity for merges on `String` fields can be changed by setting the `caseSensitive` property. This method has no effect on existing working record settings.
– **assign( )** — Called on a JSDO table reference or a `JSRecord` reference, this method updates field values for an existing record in JSDO local storage from property values of an object passed as a parameter to the method; has no effect on existing working record settings.

**Note:** An `assign( )` method is also available on a table reference of a `progress.ui.UIHelper` object. However, it updates the field values of an existing record from values in an HTML page. For more information, see the “Getting started with other HTML coding tools” section on page 111.

– **remove( )** — Called on a JSDO table reference or a `JSRecord` reference, this method deletes an existing record in JSDO local storage; no working record is set for the table reference or any of its child table references.

These methods always execute synchronously. You save the changes that these methods make in JSDO local storage to the AppServer by calling the `saveChanges( )` method on the JSDO, which executes asynchronously.

### Sorting record objects in JSDO local storage:

– **setSortFields( )** — Specifies or clears the record fields on which to automatically sort the record objects for a table reference after you have set its `autoSort` property to `true`. When enabled, this automatic sorting occurs in JSDO local storage after supported JSDO operations update the associated data.

– **setSortFn( )** — Specifies or clears user-defined sort function on which to automatically sort the record objects for a table reference after you have set its `autoSort` property to `true`. When enabled, this automatic sorting occurs in JSDO local storage after supported JSDO operations update the associated data.

– **sort( )** — Sorts the existing record objects for a table reference in JSDO local storage using either specified sort fields or a specified user-defined sort function. This function sorts the record objects for a table reference whether or not the associated data has changed.

These methods have no effect on existing working record settings, are all called on a JSDO table reference, and always execute synchronously. The case sensitivity of sorting on `String` fields can be changed by setting the `caseSensitive` property.

### Searching for record objects in JSDO local storage:

– **find( )** — Searches for a record in a referenced temp-table according to the criteria defined by a function that you pass, and returns the record object if the function indicates it has been found; sets any record found as the working record for the table reference, and sets the working record for any child table references to the first record that is related to the parent working record.

– **findById( )** — Searches for a record in a referenced temp-table with the specified record ID, and returns the record object if found; sets any record found as the working record for the table reference, and sets the working
record for any child table references to the first record that is related to the parent working record.

- `foreach()` — Loops through the records of a referenced temp-table, and allows a function that you pass to access each record object and perform whatever actions you define until the function tells the method to stop looping or the method has reached the end of the record set; the record for each iteration is set as the working record for the table reference, and sets the working record for any child table references to the first record that is related to the parent working record. If the loop terminates, the last working record remains the working record for the table reference.

These methods are called on a JSDO table reference and always execute synchronously.

- Returning data and information from JSDO local storage:
  - `getData()` — Called on a JSDO table reference, this method returns an array of record objects for a referenced temp-table; has no effect on existing working record settings.
  - `getId()` — Called on a JSDO table reference or a `JSRecord` reference, this method returns the unique internal record ID for the specified record object; has no effect on existing working record settings.
  - `getSchema()` — Called on a JSDO table reference, this method returns an array of objects, one for each field, that defines the schema of the referenced temp-table; has no effect on existing working record settings.

These methods always execute synchronously.

- Managing JSDO event subscriptions:
  - `subscribe()` — Subscribes a given event handler function to a named event on a JSDO or on a JSDO table reference; has no effect on existing working record settings.
  - `unsubscribe()` — Unsubscribes a given event handler function from a named event on a JSDO or on a JSDO table reference; has no effect on existing working record settings.

These methods are called on the JSDO or on a JSDO table reference, and execute synchronously. For more information on JSDO events and managing JSDO event subscriptions, see the “Asynchronous and synchronous execution” section on page 87.
Asynchronous and synchronous execution

As described in the previous section, most JSDO methods execute synchronously, which means that the Mobile App waits for the method to complete execution before executing the next JavaScript statement or evaluating the next term of an expression. When method execution completes and returns its results, the Mobile App continues with the next JavaScript operation.

However, the JSDO `fill()` and `saveChanges()` methods, which execute built-in Mobile operations, always execute asynchronously, and any non-built-in invocation methods, which execute non-built-in Mobile invoke operations, execute asynchronously by default, but can optionally execute synchronously. Asynchronous execution means that immediately after the Mobile App calls the method, it continues to execute the next JavaScript statement or to evaluate the next term of an expression. Results of an asynchronous method execution only become available to the Mobile App when an event associated with the method fires and an event handler function that is subscribed to the event executes and receives the results.

Comparing asynchronous and synchronous execution

Asynchronous execution is mandatory for the methods that execute the built-in Mobile CRUD operations. These operations usually involve AppServer access to its data sources, which are typically OpenEdge databases. The Mobile read operation executed by the `fill()` method can involve reading and returning hundreds to thousands (or more) records in multiple temp-tables of a ProDataSet across the network. The Mobile create, update, and delete operations, which require writes to and lock management of databases, are executed across the network by the `saveChanges()` method one record at a time, and for as many records as are marked as changed in JSDO local storage. This means that completion of these methods can require detectable wait times. If they were executed synchronously, the Mobile App user would be prevented from doing any work within the app while these methods were executing. With asynchronous execution, they can perform other tasks, such as setting application options, while waiting for a list of customers and orders, for example, to be displayed in a list view, or while waiting for notification that changes they have posted have been saved in the database. For more information on the execution options for methods that execute the built-in Mobile CRUD operations, see the “Accessing built-in CRUD operations” section on page 95.

Asynchronous execution is the default for non-built-in invocation methods, again, because these operations execute across the network and can involve complex database interactions on the AppServer. However, it is also possible that an invocation method might perform a simple function and return a primitive value used in an expression—for example, an age or credit limit. In this case, you might prefer to execute the method synchronously in the expression to complete the calculation with its return value. This you can do by passing `false` as an additional `Boolean` parameter that indicates that the invocation method is to be executed synchronously. For more information on the execution options for invocation methods, see the “Accessing non-built-in invoke operations” section on page 104.
Named events for asynchronous execution

For all methods that execute asynchronously, the JSDO supports a set of named events to which you can subscribe _event handler functions_, functions that execute with a signature defined to receive the results of a particular event. The JSDO defines a set of unique before and after events for each Mobile operation and also for the `saveChanges( )` method, itself, because in one call, this method can execute all three of the create, update, and delete operations for any number of records in JSDO local storage.

In other words, there is a unique event that fires _before_ a given Mobile operation executes across the network and one that fires _after_ a given Mobile operation executes. In addition, there is one event that fires _before_ the `saveChanges( )` method executes any Mobile operations and another that fires _after_ all Mobile operations executed by the `saveChanges( )` method have completed. Note that the after events all fire (including for `saveChanges( )`) whether Mobile operations complete successfully or with an error.

So, each asynchronous method has a set of unique `'before*' and `'after*' events defined to fire for it as follows:

- **fill( )**:
  - `'beforeFill'` — Fires before the Mobile read operation executes
  - `'afterFill'` — Fires after the Mobile read operation completes execution

- **saveChanges( )**:
  - `'beforeSaveChanges'` — Fires before the method executes any Mobile operations
  - `'beforeDelete'` — Fires before any Mobile delete operation executes
  - `'afterDelete'` — Fires after each Mobile delete operation completes execution
  - `'beforeCreate'` — Fires before any Mobile create operation executes
  - `'afterCreate'` — Fires after each Mobile create operation completes execution
  - `'beforeUpdate'` — Fires before any Mobile update operation executes
  - `'afterUpdate'` — Fires after each Mobile update operation completes execution
  - `'afterSaveChanges'` — Fires after all the Mobile operations executed by the method have completed

**Note:** These events are listed in general firing order for a single call to the `saveChanges( )` method. When there are multiple changes to the same record, the order and number of changes is optimized to send the fewest number of operations to the AppServer.
Non-built-in invocation methods:

- 'beforeInvoke' — Fires before the specified Mobile invoke operation executes
- 'afterInvoke' — Fires after the specified Mobile invoke operation completes execution

**Note:** When you subscribe an event handler to an invoke event, you also specify the name of the JSDO invocation method so the event handler can identify which invoke operation is about to be, or has been, executed. Note also that these invoke events never fire when you execute an invocation method synchronously because all invoke operation results are returned by the method itself.

**Note:** The JSDO event names are quoted because they are always referenced as strings in JavaScript.

Managing JSDO event subscriptions

You can subscribe event handlers to JSDO events using either the `subscribe()` method on the JSDO or by setting appropriate properties in an initialization object that you can pass to the constructor to instantiate the JSDO. If you use the `subscribe()` method after the JSDO is instantiated and its local storage has been loaded with records, you can also subscribe to events for the Mobile create, update, and delete operations that execute only for a specific table reference.

When you subscribe an event handler function to a JSDO event, the parameter list for the function must match the parameter list defined for the event. However, every event handler receives a reference to the JSDO as its first parameter and a reference to a request object as its last parameter that contains event results (see the following section). All handlers for 'after*' events receive a Boolean parameter that indicates the success of the Mobile operation (or operations for 'afterSaveChanges'). All handlers for events fired by Mobile create, update, and delete operations receive a JSRecord object parameter that represents the record object in local storage that is created, updated, or deleted. For more information on the required parameter list for each event, see the reference entry for the event in Appendix C, “OpenEdge JavaScript Properties, Methods, and Events Reference.”

Regardless of how you subscribe event handlers to an event, you can remove an event subscription for an event handler using the `unsubscribe()` method. If an event has no event handler subscribed and the event fires, it returns no results to the Mobile App.

Handling asynchronous and synchronous execution results

For the synchronous JSDO methods (such as `add()` or `find()`) that do not execute Mobile operations, the results for both successful and unsuccessful execution are as defined for each method. For more information, see the reference entry for the method in Appendix C, “OpenEdge JavaScript Properties, Methods, and Events Reference.”
For methods that execute Mobile operations asynchronously (\( \text{fill}() \), \( \text{saveChanges}() \), and invocation methods), the results for each event are returned in a general request object that is passed as the last parameter to the event handler. For invocation methods that you execute synchronously, a reference to this same request object is available as the return value of the method. This request object has a number of properties whose values depend on the event. Some properties (including the jsdo, jsrecord, and success properties) duplicate the settings of the other parameters passed to the event handler. The settings of other properties provide additional information appropriate for the event.

One of the most important is the response property of the request object. This property is set only for the 'after*' events of all Mobile operations (that is, all except the 'afterSaveChanges' event). It references a JavaScript object that a Mobile operation returns for a successful completion or with ABL errors.

For a built-in Mobile operation that completes successfully, this property references a JavaScript object that contains the data of the returned temp-table or ProDataSet converted from any valid JSON returned for the operation over the network, and is otherwise null.

For a non-built-in invoke operation that completes successfully, the response property references an object that contains properties with the values of any output parameters and return value returned by the ABL routine. A property for an ABL output parameter has the same name (and letter case) as the corresponding output parameter, and has a JavaScript data type that maps to the ABL data type of the output parameter (see Appendix A, "ABL to JavaScript Data Type Mapping"). Any return value from the routine is returned as the OpenEdge-defined property, _retVal, also with a ABL-mapped JavaScript data type.

For a Mobile operation that completes with one or more ABL errors, the response property references an object that contains two OpenEdge-defined properties:

- **_retVal** — A String with the value of any ABL RETURN ERROR string or ReturnValue property for a thrown AppError object
- **_errors** — An array of JavaScript objects, each of which contains properties with the ABL error message string and error number for one of possibly many ABL-returned errors.

**Note:** In the current OpenEdge release, this array always returns one object only for the first ABL error (the equivalent of ERROR-STATUS:GET-MESSAGE(1) in ABL).

For more information on the request object and its available properties, see its reference entry in Appendix B, “OpenEdge JavaScript Class and Object Reference”.

**General error handling for Mobile operations**

For any Mobile operation that completes with an error of any kind (ABL, Mobile Web application, or network), the success property of the returned request object (and the success parameter of the event handler) is set to false. As noted in the previous section, any ABL errors can be found in the object returned by the response property. All other Web application and network errors can be inspected using the xhr property of the request object. For more information on the XMLHttpRequest object that this property references, see the software development documentation for your Web browser or mobile device.
Note that for a Mobile create, update, or delete operation, if any error, at any point causes the operation to complete unsuccessfully, the record in JSDO local storage is reverted prior to any change that caused the operation to be executed. So, for example, a failed create operation causes the added record to be removed from local storage. The record object that was originally added for the operation is then available for your re-use as the jsrecord property of the request object (or the record parameter if the event handler). A similar reversion occurs for the update and delete operations, with the field values of any updated record reverted to their original values, and any deleted record added back to local storage.

**Properties of a JSDO**

Every JSDO has several properties to manage its state. We have already introduced the table reference properties that provide access to temp-table data loaded into JSDO local storage, and the record property, which provides access to individual record objects on a given table reference (see the “How JSDO local storage works” section on page 81).

Four additional properties are available for access on a JSDO:

- **autoSort** — A Boolean property on a JSDO and its table references that indicates if record objects are sorted automatically at the completion of a supported JSDO operation that updates local storage. In addition to setting this property to `true` (the default), in order to enable automatic sorting, you must invoke one or both of the `setSortFields()` and `setSortFn()` methods to specify how associated record objects are to be sorted—using specified sort fields or a user-defined sort function, respectively. You can set and reset this property at any time during the life of a JSDO to change what table references in local storage are automatically sorted.

- **caseSensitive** — A Boolean property on a JSDO and its table references that indicates if comparisons of `String` fields performed by supported JSDO operations on record objects in local storage are case sensitive or case-insensitive for the affected table references. The setting of this property affects `String` comparisons only for merging record objects into JSDO local storage using the `addRecords()` method and for the sorting of record objects in JSDO local storage, including automatic sorting using the `autoSort` property and manual sorting using the `sort()` method. You can set and reset this property at any time during the life of a JSDO to change the case sensitivity of these supported `String` comparisons for selected table references in local storage. However, note that any default `String` comparisons that you perform directly in JavaScript are case sensitive following JavaScript rules, regardless of the `caseSensitive` property setting.
Chapter 4: Creating Mobile Apps using JSDOs

- **name** — A **String** property that returns the name of the Mobile resource for which the JSDO is created. You must set this value either directly or indirectly using the `progress.data.JSDO` class constructor when you instantiate a JSDO. You can read the property on the JSDO after it is created.

- **useRelationships** — A **Boolean** property that when set to `true` makes all data-relations active in JSDO local storage so that searches on a child temp-table reference involve only records related to a its parent. In addition, record objects created for a child temp-table have their key fields automatically set in relation to their parent. When set to `false`, searches on a child temp-table reference involve all record objects stored for the temp-table, regardless of data-relations, and any record objects created for a child temp-table have no automatic settings for their key fields. You can set and reset this property at any time during the life of a JSDO to change the effect of data-relations on local storage.

In addition to these JSDO properties, you can set additional initialization properties in the class constructor, along with the `name` property itself. These initialization properties, which you cannot access after the JSDO is created, allow you to specify that certain JSDO methods are called during instantiation, including:

- Automatically calling the `fill()` method to initialize JSDO local storage as the object is created
- Subscribing event handlers for JSDO events, especially the 'beforeFill' and 'afterFill' events to handle the automatic execution of the `fill()` method

For more information on setting the name and initialization properties in the JSDO class constructor, see the “Creating a JSDO for a Mobile resource” section on page 94.

**Requirements for using a JSDO**

The basic requirements for using a JSDO include having access to the Mobile resource it needs to access and set up a user login session to access that Mobile resource.

To use a JSDO in a Mobile App, you need to ensure that the following actions have been completed:

1. Deploy a Mobile Web application that contains a Mobile service with the Mobile resource, or resources, you need to an Apache Tomcat Web server (such as the OpenEdge-installed OE Web Server), whether for testing or production. For more information on deployment for testing, see the sections on publishing Mobile services for testing in Chapter 3, “Creating Mobile Services.” For more information on deployment for production, see the sections on deploying Mobile Web applications in Chapter 6, “Deploying Mobile Applications.”

2. Instantiate the `progress.data.Session` class to create a user login session between the Mobile App and the Mobile Web application and ensure that you initialize it to use the same Web server authentication model as the Mobile Web application. For more information on coding for security considerations like the Web server authentication model, see the “Managing user login sessions” section on page 106.
3. Call the `login()` method on the `Session` object to establish an authenticated login session with the Mobile Web application. Depending on the Web server authentication model, you should do this in a way that ensures your Mobile App authenticates the user against a separate Web application resource, such as an HTML page, before you attempt to access Mobile services and resources in the Web application. You can do this using features of the `login()` method itself that allow you to access a default Web resource or one of your own that you build into the Web application. For more information, see “Managing user login sessions” section on page 106.

4. Once you have successfully called the `login()` method on the `Session` object, call the `addCatalog()` method on the `Session` object to load the JSDO catalog file for a Mobile service you need. For more information, see “Managing user login sessions” section on page 106.

5. Once you have the JSDO catalog file loaded in your Mobile App, you can instantiate the `progress.data.JSDO` class to create the JSDO for the Mobile resource you need. For more information, see “Creating a JSDO for a Mobile resource” section on page 94.
Creating a JSDO for a Mobile resource

The following code fragment shows a simple JavaScript coding sequence for creating a JSDO, starting with the attempted login of the user:

```javascript
// create Session and set it for HTTP Basic Authentication
var pdsession = new progress.data.Session();
pdsession.authenticationModel = progress.data.Session.AUTH_TYPE_BASIC;

// log in, i.e., authenticate to the Mobile Web application
pdsession.login('/SportsMobile', username, password);

if pdsession.loginResult == progress.data.Session.LOGIN_SUCCESS {

    // load catalog for a service that's part of the Mobile Web application
    pdsession.addCatalog('/SportsMobile/static/mobile/SportsMobileSvc.json');

    // create JSDO
    var dsCustomer = new progress.data.JSDO({
        name : 'dsCustomer',
        autoFill : true,
        events : {
            'afterFill' : [
                { scope : this,
                  fn : function (jsdo, success, request) {
                      // afterFill event handler statements ...
                  }
            ]
        ]
    });
    // JSDO access continues ...
}

// Login error begins ...
```

The fragment shows the setting of the authenticationModel property on the Session object, pdsession, to code for the Web server HTTP Basic Authentication model for which the Mobile Web application is configured. After logging into the Mobile Web application with its relative URI, /SportsMobile, the code tests for the success of the login, and for a positive test, immediately loads the JSDO catalog for the SportsMobileSvc service. (For more information on logging in and managing user access to Mobile services, see the “Managing user login sessions” section on page 106.)

It then instantiates a JSDO using an initialization object passed to the constructor, which specifies the:

- **Mobile resource**, dsCustomer
- **Setting of the initialization property**, autoFill, to invoke the fill( ) method upon instantiation
- **Subscription of an in-line function to the afterFill event to check the results of the fill( ) method**
Accessing built-in CRUD operations

After creating a JSDO as explained in the preceding section, you can use the built-in JSDO methods to read, create, update, and delete records. The sections that follow provide guidance and examples for each of these operations. In all cases, examples are based on a JSDO having been created and associated with a Mobile resource named dsOrderEntry.

In the case of all four CRUD operations, the ABL routine that is associated with the JSDO method must have a parameter whose type is either DATASET or TABLE. For operations to work properly, this parameter must refer to the same ProDataSet or temp-table in all ABL routines associated with CRUD operations for a given JSDO. No validation of this requirement occurs on the client.

**Note:** Built-in JSDO methods execute asynchronously.

### Read operation

To load data into local storage on the client, you call the `fill()` method on the JSDO. Each time `fill()` is called, all records currently in local storage are cleared and replaced by the records returned by the method.

When the operation is complete, the working record for each referenced temp-table is set to its first record, depending on any active parent-child relationships. So, for each child temp-table, the first record is determined by its relationship to the related working record in its parent temp-table.

This method executes the ABL method or user-defined function that is associated in the JSDO catalog with the read operation.

**Associated ABL routine: Read**

The ABL routine associated with a read operation must have the following signature:

- An input parameter of type CHARACTER, named `filter`. This parameter is optional for the `fill()` call that initiates the read operation; if specified, it defines the criteria for a filtered subset of records to be returned. The format and content of the filter string depend on the application.

- A DATASET or TABLE output parameter.
The following example shows a `ReadOrderEntry()` method that might be associated with Mobile read operations:

```plaintext
METHOD PUBLIC VOID ReadOrderEntry(  
   INPUT filter AS CHARACTER,  
   OUTPUT DATASET dsOrderEntry):  

   DEFINE DATA-SOURCE srcCustomer FOR Customer.  

   EMPTY TEMP-TABLE ttCustomer.  

   IF filter NE "" AND filter NE ? THEN  

   DATASET dsOrderEntry:FILL( ).  
   BUFFER ttCustomer:DETACH-DATA-SOURCE( ).  
   RETURN.  
END METHOD.
```

**Client JavaScript code: Read**

The following example illustrates calling `fill()` on the JSDO to load records from the database into local storage:

```plaintext
var strFilter = 'where CustNum < 100';  
/* subscribe to event */  
dsOrderEntry.subscribe('afterFill', onAfterFill);  
dsOrderEntry.fill(strFilter);  

function onAfterFill(jsdo , success , request ) {  
   if (success) {  
      /* for example, add code to display all records on a list */  
      jsdo.foreach(function (jsrecord) {  
         /* the code here is executed for each record on the table.  
         You can reference the fields as jsrecord.data.field */  
      });  
   } else {  
      if (request.response && request.response._errors &&  
         request.response._errors.length > 0){  
         var lenErrors = request.response._errors.length;  
         for (var idxError=0; idxError < lenErrors; idxError++) {  
            var errorEntry = request.response._errors[idxError];  
            var errorMsg = errorEntry._errorMsg;  
            var errorNum = errorEntry._errorNum;  
            /* handle error */  
         }  
      }  
   }  
};
```

Notice that the sample code:

- Passes a filter parameter, `strFilter`, to the `fill()` method. This filter causes the method to load only those records with a `CustNum` value lower than 100.
- Subscribes an event-handler function, `onAfterFill`, to the `afterFill` event to enable processing and error-checking of the returned records.
Create operation

To create a new record in local storage on the client, you call the `add()` method on a table reference on the JSDO. The fields of the new record are initialized with the values specified in an object passed to the method. For any fields whose values are not provided in this object, default values are taken from schema in the JSDO catalog.

When the operation is complete, the new record becomes the working record for the associated temp-table. If the temp-table has child temp-tables, the working record for these child tables is not set.

When you call `saveChanges()` to synchronize changes made by this method with the database, the AppServer executes the ABL routine that is associated in the JSDO catalog with the create operation.

**Note:** If execution on the AppServer results in changes to the record as compared with the record sent from the client (for example, the addition of a sequence value), the JSDO's local storage is automatically synchronized with the changes when the request object is returned.

If an error occurs on the server, the record is automatically deleted from the JSDO's local storage.

**Associated ABL routine: Create**

The signature of the ABL routine associated with a create operation must have a `DATASET` or `TABLE` input-output parameter.

The following example shows a `CreateOrderEntry()` method that might be associated with Mobile create operations:

```java
METHOD PUBLIC VOID CreateOrderEntry(INPUT-OUTPUT DATASET dsOrderEntry):
  END METHOD.
```
Client JavaScript code: Create

The following example illustrates calling `add()` on a table reference in the JSDO to create a record:

```javascript
/* subscribe to event */
dsOrderEntry.ttCustomer.subscribe('afterCreate', onAfterCreate);

/* some code that would add a record and save it */
var jsrecord = dsOrderEntry.ttCustomer.add( {State : 'MA'} );

<...>
dsOrderEntry.saveChanges();

function onAfterCreate (jsdo , record , success , request ) {
  if (success) {
    /* for example, get the values from the record for redisplaying */
    var myField = record.data.myField;
    <...>
  }
  else {
    if (request.response && request.response._errors &&
      request.response._errors.length > 0){
      var lenErrors = request.response._errors.length;
      for (var idxError=0; idxError < lenErrors; idxError++) {
        var errorEntry = request.response._errors[idxError];
        var errorNum = errorEntry._errorNum;
        /* handle error */
      }
      /* handle error */
    }
  }
};
```

Notice that the sample code:

- Subscribes an event-handler function, `onAfterCreate`, to the `afterCreate` event to enable error-checking and manipulation of the new record.

- Adds the record to the `ttCustomer` table, with an initial value of `MA` for the `State` field. (The table reference could be omitted if `ttCustomer` were the only temp-table in the `dsOrderEntry ProDataSet`.)

- Calls `saveChanges()` to execute `CreateOrderEntry()` on the AppServer and thereby synchronize the content of local storage with the database.

Update operation

To modify an existing record in local storage on the client, you call the `assign()` method. The values of fields to be updated are specified in an object passed to the method.

The `assign()` method can be called on either of the following:

- A table reference on the JSDO. When called on a table reference, as in the example below, `assign()` operates on the working record for that table.

- A specific `JSRecord` object.
When the operation is complete, any working records previously set before the method executed remain as the working records.

When you call `saveChanges()` to synchronize changes made by this method with the database, the AppServer executes the ABL routine that is associated in the JSDO catalog with the update operation.

**Note:** If execution on the AppServer results in changes to the record as compared with the record sent from the client (for example, an update caused by a database trigger), the JSDO's local storage is automatically synchronized with the changes when the request object is returned.

If an error occurs on the server, the record automatically reverts to its original state in the JSDO's local storage.

**Associated ABL routine: Update**

The signature of the ABL routine associated with an update operation must have a `DATASET` or `TABLE` input-output parameter.

The following example shows an `UpdateOrderEntry()` method that might be associated with Mobile update operations:

```abl
METHOD PUBLIC VOID UpdateOrderEntry(INPUT-OUTPUT DATASET dsOrderEntry):
END METHOD.
```
Client JavaScript code: Update

The following example illustrates calling `assign()` on a table reference in the JSDO to update a record:

```javascript
/* subscribe to event */
dsOrderEntry.subscribe('afterUpdate', onAfterUpdate);

var updatedDataObject = {City: 'Nashua'};
/* some code that would update a record and send it to the server */
var jsrecord = dsOrderEntry.ttCustomer.findById(myid);
dsOrderEntry.ttCustomer.assign( updatedDataObject );
dsOrderEntry.saveChanges();

function onAfterUpdate (jsdo , record , success , request ) {
  if (success) {
    /* for example, get the values updated by the server from the record
to redisplay */
    var newValue = record.data.myField;
    <...>
  }
  else {
    if (request.response && request.response._errors &&
        request.response._errors.length > 0){
      var lenErrors = request.response._errors.length;
      for (var idxError=0; idxError < lenErrors; idxError++) {
        var errorEntry = request.response._errors[idxError];
        var errorMsg = errorEntry._errorMsg;
        var errorNum = errorEntry._errorNum;
        /* handle error */
      }
    }
  }
}
```

Notice that the sample code:

- **Subscribes an event-handler function**, `onAfterUpdate`, to the `afterUpdate` event to enable error-checking and manipulation of the revised record.
- **Calls `findById()` to retrieve the record to be updated**, which becomes the working record for the `ttCustomer` table, and changes the value of the `City` field to Nashua. (The table reference could be omitted if `ttCustomer` were the only temp-table in the `dsOrderEntry` ProDataSet.)
- **Calls `saveChanges()` to execute `UpdateOrderEntry()` on the AppServer and thereby synchronize the content of local storage with the database.**

Delete operation

To delete an existing record from local storage on the client, you call the `remove()` method.

The `remove()` method can be called on either of the following:

- A table reference on the JSDO. When called on a table reference, `assign()` operates on the working record for that table.
- A specific `JSRecord` object. The example below uses this technique.
When the operation is complete, any working record for an associated temp-table and for any child temp-tables is not set.

When you call `saveChanges()` to synchronize changes made by this method with the database, the AppServer executes the ABL routine that is associated in the JSDO catalog with the delete operation.

**Note:** If an error occurs on the server, the record is automatically restored in the JSDO’s local storage.

**Associated ABL routine: Delete**

The signature of the ABL routine associated with a delete operation must have a `DATASET` or `TEMP-TABLE INPUT-OUTPUT` parameter.

The following example shows a `DeleteOrderEntry()` method that might be associated with Mobile delete operations:

```ABL
METHOD PUBLIC VOID DeleteOrderEntry(INPUT-OUTPUT DATASET dsOrderEntry):
END METHOD.
```

**Client JavaScript code: Delete**

The following example illustrates calling `remove()` on a `JSRecord` object:

```javascript
/* subscribe to event */
dsOrderEntry.subscribe('afterDelete', onAfterDelete);

var jsrecord = dsOrderEntry.ttCustomer.findById(myid);
jsrecord.remove();
dsOrderEntry.saveChanges();

function onAfterDelete (jsdo , record , success , request ) {
  if (success) {
    /* for example, get the values from the record that was 
     * deleted to display a confirmation message */
    var myKeyField = record.data.myKeyField;
    
    /* handle error */
  } else {
    if (request.response && request.response._errors &&
        request.response._errors.length > 0){
      var lenErrors = request.response._errors.length;
      for (var idxError=0; idxError < lenErrors; idxError++) {
        var errorEntry = request.response._errors[idxError];
        var errorMsg = errorEntry._errorMsg;
        var errorNum = errorEntry._errorNum;
        /* handle error */
      }
    }
  }
};
```
Notice that the sample code:

- Subscribes an event-handler function, `onAfterDelete`, to the `afterDelete` event to enable error-checking and manipulation of the revised record.

- Calls `findById()` to retrieve the record to be deleted, and then calls `remove()` on this JSRecord object.

- Calls `saveChanges()` to execute `DeleteOrderEntry()` on the AppServer and thereby synchronize the content of local storage with the database.
Calling saveChanges() for multiple CRUD operations

As an alternative to calling `saveChanges()` to update the database after each discrete CRUD operation, you can have OpenEdge Mobile apply all pending changes with a single `saveChanges()` call. The following example illustrates this approach:

```javascript
/* subscribe to event */
dsOrderEntry.subscribe('afterSaveChanges', onAfterSaveChanges);
dsOrderEntry.saveChanges();

function onAfterSaveChanges(jsdo, success, request) {
   /* number of operations on batch */
   var len = request.batch.operations.length;

   if (success) {
      /* all operations in batch succeeded */
      /* for example, redisplay records in list */
      jsdo.foreach( function(jsrecord) {
         /* reference the record/field as jsrecord.data.fieldName */
      });
   } else {
      /* one or more operations in batch failed */
      for(var idx = 0; idx < len; idx++) {
         var operationEntry = request.batch.operations[idx];

         console.log("Operation: " + operationEntry.fnName);
         var opName;
         switch (operationEntry.operation) {
            case progress.data.JSDO._OP_CREATE:
               opName = 'create';
               break;
            case progress.data.JSDO._OP_UPDATE:
               opName = 'update';
               break;
            case progress.data.JSDO._OP_DELETE:
               opName = 'delete';
               break;
         }

         if (!operationEntry.success) {
            /* handle error condition */
            if (operationEntry.response &&
               operationEntry.response._errors &&
               operationEntry.results._errors.length > 0) {
               var lenErrors = operationEntry.response._errors.length;
               for (var idxError=0; idxError < lenErrors; idxError++) {
                  var errors = operation.results._errors[idxError];
                  var errorMsg = errors._errorMsg;
                  var errorNum = errors._errorNum;
                  /* handle error */
               }
            }
         } else {
            /* operation succeed */
         }
      }
   }
};
```
Accessing non-built-in invoke operations

In addition to the built-in CRUD methods, a JSDO can call invocation methods that correspond to ABL routines defined in the Mobile resource and annotated with an "invoke" operation type. The JSDO catalog identifies the available invoke operations. Calling an invoke operation on the JSDO causes the corresponding routine to execute on the AppServer.

The signature of an ABL routine designated as an invoke operation is unrestricted. Parameters of ABL data types supported by OpenEdge Mobile can be used.

The invocation method name can be the same as that of the ABL routine, or it can be an alias, as defined by the resource. The method passes any ABL input parameters as properties of an object parameter. The method returns results from the ABL routine, including any return value and output parameters, in the response property of a request object.

The response property is an object in which parameter names match the names defined in the ABL routine. Since JavaScript is case-sensitive, code that accesses the value of an output parameter must exactly match the name defined in the ABL routine.

For user-defined functions and non-void ABL methods, the return value is available in the _retVal property of the response object. The _retVal property also contains any error information returned by the server if the request fails.

Note: The JSDO local storage is not automatically updated with the results of an invoke operation. To add records returned by the invoke operation to local storage, call addRecords( ) on the appropriate table.

Asynchronous vs. synchronous method execution

Invocation methods can execute either asynchronously (the default behavior) or synchronously. An optional Boolean "async flag," provided as a second parameter to the method (in addition to the Object parameter, if any, containing ABL input parameters), specifies the execution mode. The value of this second parameter defaults to true if omitted. For synchronous execution, set the flag to false.

To process the results of an asynchronous method call, you subscribe an event-handler function to the afterInvoke event. For a synchronous call, you do not subscribe to the event, but access the data through the response property of the request object returned by the method.

Invocation method example

To illustrate working with different data types, the following example involves an invoke operation that both returns a DECIMAL value and has an output parameter of type TABLE.
ABL invoke routine: GetCreditLimit()

The following example shows a method that might be designated as an invoke method, which can be called on the JSDO:

```abl
METHOD PUBLIC DECIMAL GetCreditInfo ( INPUT piCustNum AS INT, OUTPUT TABLE eTable):
END METHOD.
```

Client JavaScript code: Invoking GetCreditLimit()

The following example illustrates an asynchronous call to the preceding method:

```javascript
dsOrderEntry.subscribe('afterInvoke', 'GetCreditInfo',
onAfterInvokeGetCreditInfo);
dsOrderEntry.GetCreditInfo ( { piCustNum : 10 } );

function onAfterInvokeGetCreditInfo (jsdo , success , request ) {
    var res = request.response;
    if (success) {
        var creditLimit = res._retVal;
        var eTableObj = res.eTable.eTable;
    } else {
        if (res && res._errors &&
            res._errors.length > 0){
            var lenErrors = res._errors.length;
            for (var idxError=0; idxError < lenErrors; idxError++ ) {
                var errorEntry = res._errors[idxError];
                var errorMsg = errorEntry._errorMsg;
                var errorNum = errorEntry._errorNum;
                /* handle error */
            }
        }
    }
};
```

Notice that the sample code:

- Subscribes an event-handler function, `onAfterInvokeGetCreditInfo`, to the `afterInvoke` event to enable error-checking and processing of the results.

- Declares the `eTableObj` variable with a value of `request.eTable.eTable`. The parameter name (`eTable`) must be specified twice. The first instance refers to the parameter name as defined in the ABL routine; the second instance is for the temp-table serialized as a JSON object.
Managing user login sessions

The first task in accessing a Mobile resource with a JSDO is to create a user login session for the Mobile Web application that provides the Mobile service and resource you need. This might require user authentication prior to calling the `login()` method on an instance of the OpenEdge JavaScript class, `progress.data.Session`. Once user authorization is obtained, you must then call the `addCatalog()` method on this `Session` object to load the JSDO catalog for the service into your Mobile App. You can then create a JSDO for a Mobile resource defined in the catalog.

However, you need to gather some information to determine how best to configure and code the user login sequence.

Requirements for creating a user login session

You need to identify the following information to configure and code a user login session:

- **The Web server authentication model that the Mobile Web application will use:**

  Web servers support a number of authentication models to manage client access to Web resources. OpenEdge Mobile supports the following authentication models:

  - **Anonymous** — No authentication is required. This is the default value.
  
  - **HTTP Basic Authentication** — The Mobile Web application requires a valid user ID and password, but does not provide a page containing a login form (credentials are typically entered in a generic login dialog provided by either the Mobile App, the Web browser, or the native device container in which the App is running).
  
  - **HTTP Forms Authentication** — The Mobile Web application requires a valid user ID and password and provides a page containing a login form.

  The exact user login sequence depends on the Web server authentication model, the Mobile App type and platform, and how Mobile Web application resources are protected on the Web server. Because of this, differences in platform behavior can cause problems with reliable access to Mobile resources. To minimize the chances of such access problems, you must know the Web server authentication model to be configured for the Mobile Web application and set the `authenticationModel` property on the `Session` object accordingly. For more information on Web server authentication models and OpenEdge Mobile, see the sections on security considerations in Chapter 6, “Deploying Mobile Applications.”
Managing user login sessions

- **Whether the Web browser(s) or mobile device(s) where the Mobile App runs will have cookies enabled:**

  If the Mobile Web application you access will use HTTP Form Authentication, the mobile devices and Web browsers that access the Mobile Web application **must** have cookies enabled. Otherwise, the Mobile App cannot login and access Mobile services. If there is any question about the availability of cookies on client platforms, you might consider using HTTP Basic Authentication for the Mobile Web application, instead, and set the `authenticationModel` property on the `Session` object accordingly.

  If the Mobile Web application will use HTTP Basic Authentication and the mobile devices and Web browsers will not have cookies enabled, you **must** set a property in the single sign-on (SSO) configuration of the Mobile Web application to allow session logins to work from the Mobile App. For more information, see the sections on enabling SSO for a Web application in Chapter 6, “Deploying Mobile Applications.”

- **A protected Web resource, such as a Web page, provided by the Mobile Web application against which your Mobile App can authenticate before it tries to access its first protected Mobile resource:**

  In general, you need to be sure that security is configured to complete authentication before a Mobile App requests resources in the JSDO catalog. Although it is possible to configure Web application security so that the Mobile resources in the catalog are the only resources that require authentication, Progress Software does not recommend this approach. In certain situations where a Mobile resource is the first Web application resource to require user authentication, a Mobile App can be prevented from accessing the resource even when user authentication succeeds. Instead, Progress Software recommends that you require user authentication to access at least one Web application resource in addition to those defined in the catalog, and require that this user access and authentication occur prior to accessing any Mobile resources in the catalog. Once the user is authenticated in this way, the Web server provides access to all other resources of the Mobile Web application, including catalog resources, according to the user’s authorization settings.

  **Note:** An OpenEdge Mobile Web application includes default features to support this approach to authenticating users before any catalog resources are accessed. For more information, see the “Default Web pages to support Mobile App login” section on page 108.
The type of Mobile App you are writing and where it will be deployed: a Mobile Web App deployed to a Web server or a Mobile Native App deployed to an app store:

In order to log into a Mobile Web application and load any of its JSDO catalogs, you need to provide appropriate URIs for both. If you are writing a Mobile Web App, and it will be deployed to the same Apache Tomcat Web server as the Mobile Web application it is accessing, all of these URIs can be relative to the Web server root (domain or host and port). The Web browser automatically prepends these relative URIs to the Web server root from which the Mobile Web App is loaded. However, if a Mobile Web App is deployed to a different Web server from the one where the Mobile Web application is deployed, all of these URIs must be absolute and include the Web server root for the Mobile Web application.

If you are writing a Mobile Native App, all of the URIs for the Mobile Web application and its JSDO catalogs must also be provided as absolute URIs, because a Mobile Native App is loaded and executed from the local storage of the mobile device, which knows nothing of the Mobile Web application its Mobile Web App is going to access.

Note: In general, if a Mobile App requires absolute URIs, you need to maintain separate JavaScript sources for versions of the Mobile App that you deploy for different Mobile Web application environments, such as one for testing and another for production.

Depending on the Web server authentication model, you might need to set up Web resources differently to help with the Mobile App login sequence. OpenEdge provides some default Web resources with every deployed Mobile Web application that you can use for this purpose, or you define similar Web resources of your own.

Default Web pages to support Mobile App login

When you deploy Mobile services to a Tomcat Web server as a Mobile Web application, OpenEdge provides default Web pages that you can use to configure Mobile Web application authentication. The URIs for the following Web pages (provided relative to the Mobile Web application root) support the startup and user login of a Mobile App in a way that authenticates the user prior to requesting access to Mobile resources, depending on the Web server authentication model:

- */index.html* — Default public welcome page for Mobile App startup. This page can be protected or unprotected depending on how you design authentication for your Mobile App.

- */static/auth/login.html* — Default Mobile Web application login page to support HTTP Forms Authentication. With a login page configured (such as this default), the Mobile Web application can then provide a protected Web resource for access by the user. This might be a link on the public welcome page to a protected Web page that the user must access to use Mobile resources, or it might be the welcome page, itself, that is protected and requires authentication for access by the user.
In any case, when the user goes to the first Web application-protected Web page, the Web server returns the configured login page to the browser or mobile device, which displays the login form for the user to enter their credentials. When the user submits their credentials, the browser or device sends them to the Mobile Web application for authentication. If successful, the Web application then returns the protected Web page to the browser or device for display to the user. At this point, the user is authenticated and all further access to Mobile Web application resources is provided according to the user’s authorization settings.

The Mobile App must then call the `login()` method on a newly instantiated `Session` object, passing it the URI of the Mobile Web application to establish the login session to access its Mobile services. Since user authentication has already occurred, there is no need to pass user credentials, and any that you do pass to the `login()` method are ignored. The Mobile App is now ready to load a JSDO catalog.

An alternative approach is supported by OpenEdge Mobile for HTTP Forms Authentication, where no protected Web page or other Web resource is provided before calling the `login()` method. When this happens, after obtaining user credentials, perhaps using a Mobile App-provided login page, the method both authenticates the user directly with the HTTP Forms Authentication system and sends a request to start a login session to access JSDO catalogs.

- `/static/home.html` — Default Mobile Web application login target page to support HTTP Basic Authentication. Generally, this page is not designed to be displayed as part of Mobile App UI, but to be used only as a protected Web resource against which to authenticate the user prior to accessing Mobile resources. In this case, the Mobile App obtains the user credentials either by using a prompt directly from the browser or mobile device when the user first tries to access a protected Web page of the Mobile Web application or by using a separate login page that the Mobile App provides for the user to access and enter their credentials for authentication during session login.

If the credentials are prompted by user access to a protected Web page, the browser or device sends them to the Mobile Web application for authentication. If successful, the Web application then returns the protected Web page to the browser or device for display to the user. At this point, the user is authenticated and all further access to Mobile Web application resources is provided according to the user’s authorization settings.

The Mobile App must then call the `login()` method on a newly instantiated `Session` object, passing it the URI of the Mobile Web application to establish the login session to access its Mobile services. If the user credentials have already been prompted by the browser or device, and successfully authenticated by the Web application, there is no need to pass them, and any that you do pass to the `login()` method are ignored. The Mobile App is now ready to load a JSDO catalog.
If the credentials have been entered using a separate login page that the Mobile App provides for the user to access, you must pass them explicitly as parameters to the `login()` method in order to have them sent to the Mobile Web application for authentication together with the session login request. In this case, the Mobile Web application authenticates the user against the protected login target page, which can be the `/static/home.html` default or another protected page whose URI you pass explicitly to the `login()` method. If authentication against this login target page is successful, the user is authenticated and the login session to access Mobile services is established. All further access to Mobile Web application resources is provided according to the user’s authorization settings, and the Mobile App is now ready to load a JSDO catalog.

Again, using these suggested options for designing a login sequence, at each point where the Mobile App is ready to load a JSDO catalog, user authentication has already occurred against a protected Mobile Web application resource prior to requesting a Mobile resource.
Getting started with other HTML coding tools

If you do not need the support for the Mobile App Builder, or have other client requirements, you can also build the Mobile App using the basic HTML and JavaScript coding tools of Progress Developer Studio for OpenEdge. For this purpose, OpenEdge Mobile provides a JSDO helper class, progress.ui.UIHelper, to map JSDO data to HTML elements using supported UI frameworks, such as JQuery Mobile.

The UIHelper class assists in building Mobile applications with a list view and a detail page. It can be used in the following scenarios and JavaScript frameworks:

- JQuery Mobile
- JQuery Mobile using the Mobile App Builder
- iUI
- User interface based on HTML and JavaScript where list views are built directly using the `<li>` element

Figure 5 shows a sample screen generated using the UIHelper class.

![Sample display using the UIHelper class](image)

Using the UIHelper class

The UIHelper is instantiated to work with a JSDO instance. The `setListView()` method is used to specify the HTML element for the list (`<ul>` element). The `clearItems()` and `addItem()` methods are used to build the list view. The `showListView()` method is used to show the list view on the screen.

**Note:** For complete information on the methods of the UIHelper class, see Appendix C, “OpenEdge JavaScript Properties, Methods, and Events Reference.”
The following is a sample HTML file using JQuery Mobile (index.html):

Sample HTML file using Jquery Mobile (index.html)

```html
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <title>Customers</title>
    <link rel="stylesheet" href="http://code.jquery.com/mobile/1.1.0/jquery.mobile-1.1.0.min.css" />
    <style>
      /* App custom styles */
    </style>
    <script src="http://ajax.googleapis.com/ajax/libs/jquery/1.7.1/jquery.min.js"></script>
    <script src="http://code.jquery.com/mobile/1.1.0/jquery.mobile-1.1.0.min.js"></script>
    <script src="progress.session.js"></script>
    <script src="progress.js"></script>
    <script src="customers.js"></script>
  </head>
  <body>
    <div data-role="page" id="custlist">
      <div data-role="header">
        <h3>Customers</h3>
      </div>
      <div data-role="content">
        <ul id="listview" data-role="listview" data-divider-theme="b" data-inset="true" data-filter="true">
        </ul>
      </div>
    </div>
    <div data-role="page" id="custdetail" data-add-back-btn="true" data-theme="b">
      <div data-theme="b" data-role="header"><h1>Customer</h1></div>
      <div data-role="content">
        <form action="" id="customerform">
        </form>
      </div>
    </div>
  </body>
</html>
```
The following is the `customers.js` script used in the sample HTML.

### Sample JavaScript file using JQuery Mobile (customers.js)

```javascript
var customers;
var uihelper;
var forminitialized = false;

$(document).ready(function() {
    var session = new progress.data.Session();

    session.login("http://localhost:8980/SportsApp", "", "");

    customers = new progress.data.JSDO({ name: 'CustomerOrder' });
    customers.subscribe('AfterFill', onAfterFillCustomers, this);

    uihelper = new progress.ui.UIHelper({ jsdo: customers });

    uihelper.eCustomer.setDetailPage({ name: 'custdetail' });
    uihelper.eCustomer.setListView({
        name: 'listview',
        format: '{CustNum}<br>{Name}<br>{State}',
        autoLink: true
    });

    $('#customerform').html(uihelper.eCustomer.getFormFields()
        + '<input type="button" value="Add" id="btnAdd"/>
        + '<input type="button" value="Save" id="btnSave"/>
        + '<input type="button" value="Delete" id="btnDelete"/>
    );

    customers.fill();
});

function onAfterFillCustomers() {
    uihelper.eCustomer.clearItems();
    customers.eCustomer.foreach(function(customer) {
        uihelper.eCustomer.addItem();
    });
    uihelper.eCustomer.showListView();
}
```

The `setDetailPage( )` method is used to specify the name of the HTML element, generally a `<div>`, that represents the detail page.

The `getFormFields( )` method can be used to obtain the HTML text for the fields of the specified table reference. This HTML text is generally added to the HTML element representing the form in the detail page. This element does not need to be a `<form>` element; it can be a `<div>` element.

The format property of the initialization object for `setListView( )` contains substitution parameters that are replaced when `addItem( )` is called: `{CustNum}`, `{Name}`, and `{State}`. The working record of the specified table reference is used to determine the values of the fields. In the sample `customer.js` script, the call to `addItem( )` queries the working record to obtain the values of the fields `CustNum`, `Name` and `State`. The `addItem( )` method then builds a list item using a default template for list items.
Using a custom template

You can specify a different template for the list items using one of the following methods:

- **Using the** `itemTemplate` **property in the initialization object for** `setListView()`.  
- **Calling** `progress.ui.UIHelper.setItemTemplate()`.  
- **By not specifying the** `format` **property. In this case, the UIHelper uses the first**  
  **item element in the list view as the template for the items.**

You can also specify a different template for the fields returned when calling `getFormFields()`:

- **Specify the** `fieldTemplate` **property in the initialization object for** `setDetailPage()`.  
- **Call** `progress.ui.UIHelper.setFieldTemplate()`.  

Alternatively, you can define the layout for the detail page using HTML instead of calling `getFormFields()`.

The default item template looks like this:

```html
<li data-theme="c" data-id="{__id__}">
  <a href="#{__page__}" class="ui-link"
    data-transition="slide">{__format__}</a>
</li>
```

The default field template looks like this:

```html
<div data-role="fieldcontain">
  <label for="{__name__}">{__label__}</label>  
  <input id="{__name__}" name="{__name__}" placeholder="" value=""
      type="text" />
</div>
```

The templates use the following substitution parameters:

- **Used when building the list items by calling** `addItem()`:
  
  - `{__id__}` — The internal ID of the record  
  - `{__page__}` — The name attribute of the object passed as a parameter to `setDetailPage()`, which defines the detail page  
  - `{__format__}` — The format attribute of the object passed as a parameter to `setListView()` or (optionally) to `addItem()`, which identifies the fields to be included for each item in the list view.
• Used for the HTML text returned when calling \texttt{getFormFields( )}:
  
  – \{\_\_\name\_\}_ — The field name in a temp-table as defined by the \texttt{name} property in the catalog
  
  – \{\_\_label\_\}_ — The label of the field in a temp-table as defined by the \texttt{title} property in the catalog

The properties \texttt{itemTemplate} and the \texttt{fieldTemplate} can be used to change the template for a specific UIHelper instance.

For example:

```javascript
uiHelper.eCustomer.setListView({
    name: 'cust-listview',
    format: '{CustNum} {Name}<br>{Address}',
    autoLink: true,
    itemTemplate: '<li data-id="{\_\_id\_\}_">\{\_\_format\_\}_</li>'
});
```

```javascript
uiHelper.eCustomer.setDetailPage({
    name: 'cust-detail-page',
    fieldTemplate: '<input id="{\_\_name\_\}_"/>
');
```

The methods \texttt{progress.ui.UIHelper.setItemTemplate( )} and \texttt{progress.ui.UIHelper.setFieldTemplate( )} can be used to change the template to be used by the UIHelper for a JavaScript session.

For example:

```javascript
progress.ui.UIHelper.setItemTemplate('<li data-id="{\_\_id\_\}_">\{\_\_format\_\}_</li>');
```

```javascript
progress.ui.UIHelper.getFieldTemplate('<input id="{\_\_name\_\}_"/>
');
```
Chapter 4: Creating Mobile Apps using JSDOs

Publishing Mobile Apps for testing

When you create a Mobile project in Developer Studio, if you choose the option to publish changes immediately, as long as the OE Web Server that you have selected for a Mobile App is running, when you create or in any way update the Mobile App and its resources in Developer Studio, the Web server publishes it as a Mobile Web App. This also occurs after you save a Mobile App project in Mobile App Builder. The saved Mobile App is automatically downloaded to the Mobile Apps folder in the shared Developer Studio project. Developer Studio publishes the Mobile App from this folder as a WAR file, with the name of the Mobile App as its filename, to the selected OE Web Server for testing.

If the OE Web Server is not running when you initially create the Mobile App, or you did not select the project option to publish changes immediately, you can use the Servers view to publish the Mobile App by first starting the AppServer for the project (right-click on the AppServer and select Start). Once the AppServer is started and synchronized, you can start the OE Web Server instance for the Mobile App (right-click on the OE Web Server instance and select Start). Once the OE Web Server is started and synchronized, right-click on it and select Publish, and any Mobile App not yet published should publish.

To test a published Mobile App, you should already have coded the appropriate URIs to login to the Mobile Web application that provides the JSDO catalog for the published Mobile services you need. For more information, see the section on identifying URIs for Mobile Web applications in Chapter 3, “Creating Mobile Services.”

For information on deploying Mobile Apps for production, see Chapter 6, “Deploying Mobile Applications.”
Using JSDO Services in the Mobile App Builder

As described previously (see the “Getting started with the Mobile App Builder” section on page 78), JSDO Services are coding services that simplify access to the Mobile operations supported by a given JSDO and support the visual mapping of JSDO data to a Mobile App UI. This chapter describes how to use the JSDO Services in the Mobile App Builder, based on a simple application. It provides basic steps to create the JSDO Services and use their data mapping capabilities for Mobile operations. The steps include creation of a simple UI that includes both a list and detail page with create, update, and delete capabilities.

The following sections include:

- Setting up the sample Mobile service
- Creating the JSDO Services for the Mobile service
- Adding a login session to access the JSDO
- Building the sample UI
- Steps for using JSDO Services
- Error handling
Setting up the sample Mobile service

The chapter refers to a simple Mobile service named TempTableSvc that provides a single temp-table resource, Item. The remaining sections in this chapter assume that you have already created a Mobile App Builder project and have the Mobile service published and running. You can create this service from the ABL class and include file shown in this section. This is the content for the class file, TempTableBE.cls:

TempTableBE.cls

```abl
@openapi.openedge.export FILE(type="REST", executionMode="singleton", useReturnValue="false", writeDataSetBeforeImage="false").
@progress.service.resource FILE(name="Item", URI="/Item", schemaName="ttItem", schemaFile="SportsApp/AppServer/ttDefs.i").
USING Progress.Lang.*.

BLOCK-LEVEL ON ERROR UNDO, THROW.

CLASS TempTableBE:

{ttDefs.i}
DEFINE DATA-SOURCE data-srcTempTable FOR Item.

CONSTRUCTOR PUBLIC TempTableBE ():
    SUPER ()..
END CONSTRUCTOR.

@openapi.openedge.export(type="REST", useReturnValue="false", writeDataSetBeforeImage="false").
@progress.service.resourceMapping(type="REST", operation="read", URI="/filter={filter~}", alias="", mediaType="application/json").
METHOD PUBLIC VOID ReadTempTable( INPUT filter AS CHARACTER,
                                      OUTPUT TABLE ttItem ):
    DEFINE VARIABLE cWhere AS CHAR NO-UNDO.
    EMPTY TEMP-TABLE ttItem.
    BUFFER ttItem:ATTACH-DATA-SOURCE(DATA-SOURCE data-srcTempTable:HANDLE).
    cWhere = TRIM(filter).
    IF cWhere NE "" AND cWhere NE ? THEN DO:
        IF INDEX (cWhere, ")") > 0 THEN
            cWhere = "where " + cWhere.
        ELSE DO:
            IF NOT cWhere BEGINS "Where" THEN
                cWhere = SUBSTITUTE('Where Item.ItemNum = &1', cWhere).
            END.
        DATA-SOURCE data-srcTempTable:FILL-WHERE-STRING = cWhere.
    END.
    DATASET dsItem:FILL().
    BUFFER ttItem:DETACH-DATA-SOURCE().
END METHOD.
```
TempTableBE.cls

```plaintext
@openapi.openedge.export(type="REST", useReturnValue="false", writeDataSetBeforeImage="false").
@progress.service.resourceMapping(type="REST", operation="create", URI="", alias="", mediaType="application/json").
METHOD PUBLIC VOID CreateTempTable( INPUT-OUTPUT TABLE ttItem ):
    BUFFER ttItem:ATTACH-DATA-SOURCE(DATA-SOURCE data-srcTempTable:HANDLE).
    FOR EACH ttItem.
        BUFFER ttItem:MARK-ROW-STATE(ROW-CREATED).
    END.
    FOR EACH bItem:
        BUFFER bItem:SAVE-ROW-CHANGES(1, "ItemNum")
    END.
    BUFFER ttItem:DETACH-DATA-SOURCE().
    RETURN.
END METHOD.

@openapi.openedge.export(type="REST", useReturnValue="false", writeDataSetBeforeImage="false").
@progress.service.resourceMapping(type="REST", operation="update", URI="", alias="", mediaType="application/json").
METHOD PUBLIC VOID UpdateTempTable( INPUT-OUTPUT TABLE ttItem ):
    BUFFER ttItem:ATTACH-DATA-SOURCE(DATA-SOURCE data-srcTempTable:HANDLE).
    FOR EACH ttItem.
        BUFFER ttItem:MARK-ROW-STATE(ROW-MODIFIED).
    END.
    FOR EACH bItem:
        BUFFER bItem:SAVE-ROW-CHANGES(1, "ItemNum")
    END.
    BUFFER ttItem:DETACH-DATA-SOURCE().
    RETURN.
END METHOD.
```
Although the Item resource passes a single temp-table between the AppServer and the Mobile App, it relies on a ProDataSet on the AppServer to manage access to the sports2000 database for its Mobile operations. For more information on defining and publishing this sample Mobile service, see Chapter 3, “Creating Mobile Services.”
Creating the JSDO Services for the Mobile service

Creating JSDO Services for a Mobile service includes these basic tasks:

1. Load the JSDO catalog for the Mobile service.
2. Select the Mobile resource (or resources) for which you want to create a JSDO.
3. Update the JSDO Service settings for each resource.

The following sections describe these tasks for the sample Mobile service.

Loading the JSDO catalog for the Mobile service

To load the catalog for the TempTableSvc service:

1. Select Create New→Service, as shown in the following figure.

![Figure 6: Creating JSDO Services for a Mobile service](image)

2. After the Create new service dialog box appears, select JSDO Service, then click Upload a file. A dialog box displays to choose the catalog file for a given Mobile service.

3. After you choose the catalog file, you return to the Create new service dialog box.

Selecting the Mobile resource from the JSDO catalog

To select the resource to use from the TempTableSvc.json catalog file:

1. From Create new service dialog box, click Select resources. The Select resources dialog box appears. This displays all the resources supported by the
service, where each resource is listed as ServiceName.ResourceName. In this case, there is only one Item resource listed for the TempTableSvc service, listed as TempTableSvc.Item.

2. Select the resources (in this case, only TempTableSvc.Item) that you want to import into the project, and click Create services, as shown in the following figure.

![Select resources](image)

**Figure 7:** Selecting a Mobile resource for which to create JSDO Services

On the left panel, under the Services node this displays the list of JSDO Services created for the selected resource. The JSDO Services created for a given resource appear under a ServiceName.ResourceName node (in this case, TempTableSvc.Item). This is all based on the information loaded from the catalog, as shown in the following figure.

![PROJECT](image)

**Figure 8:** JSDO Services created for a selected Mobile resource

### Updating the JSDO Service settings for the resource

After loading the JSDO catalog for a Mobile service and selecting a resource, the first JSDO Service to access for that resource is the Settings Service. The Settings JSDO Service allows you to specify the run-time relative or absolute URIs to locate the JSDO catalog and the Mobile Web application that provides the Mobile service. The Settings
JSDO Service starts with default values, which you need to update for the selected resource.

To update the JSDO Service settings for the selected resource:

1. Click on the Settings service that is created for the resource you want to use (for example, `TempTableSvc_Item_Settings`, as shown in the figures). The right panel appears as follows:

![Image of Settings service panel]

2. Edit the default values for the `catalogURL` and `serviceURI` fields to use the base URI for the Mobile Web application, depending on whether you are deploying and testing on the same machine. For example, assuming that the Mobile Web application is named `TempTableSvc` and is defined in Progress Developer Studio for OpenEdge with the default URI:

   - If you are deploying and testing on the same machine, prepend the relative URI, `'/TempTableSvc'`, to the value in the `catalogURI` field and set the `serviceURI` field to the same relative URI (`'/TempTableSvc'`).
   - If you are deploying and testing from a device, or from the Mobile App Builder outside of Developer Studio (in an external browser), prepend the absolute URI (for example, `http://machine:8980/TempTableSvc`) to the value in the `catalogURI` field and set the `serviceURI` field to the same absolute URI.

   For example, these are the final values using the relative URI:

   - `catalogURI`:
     
     ```plaintext
     /TempTableSvc/static/mobile/TempTableSvc.json
     ```
   - `serviceURI`:
Chapter 5: Using JSDO Services in the Mobile App Builder

These are the final values using the absolute URI:

- **catalogURI:**
  
  ```
  http://localhost:8980/TempTableSvc/static/mobile/TempTableSvc.json
  ```

- **serviceURI:**
  
  ```
  http://localhost:8980/TempTableSvc
  ```

3. Once you are done, click **Save** on the top of the Mobile App Builder window and close the Settings JSDO Service tab for this resource.
Adding a login session to access the JSDO

To be able to use a JSDO, you have to create a `progress.data.Session` object and call its `login()` and `addCatalog()` methods to create a login session and load the JSDO catalog with which to instantiate the JSDO. The created JSDO Services do not do this for you. You can add the code to do this by manually adding a handler for the Load event of an appropriate page.

Adding the login code manually

To add a Load event handler to start a login session for the home page of the example:

1. Click on the **home** page, under the **Pages** node in the left panel, as shown in the following figure:

   ![Selecting events for a page](image)

   **Figure 10: Selecting events for a page**

   2. Click on **Events** under the phone frame. The **Component** displays as **home**.

   3. Select **Load** as the event and **Run Javascript** as the **Action**. This displays a small box where you can enter code.
Chapter 5: Using JSDO Services in the Mobile App Builder

Note: You can make this box wider by clicking on the button, or maximize it with the button.

Figure 11: Adding login code for a new Load event

4. You can enter code similar to the following that instantiates the Session object and makes the login() and addCatalog() calls, as appropriate. Note that it does not have code for handling a connection or login failure. It is a simple example with no user authentication. (For sample login code with user authentication, see the “Login code with user authentication” section on page 127). The comment in bold explains what you need to change based on your particular Mobile resource:

Login code without user authentication

```javascript
/* if the first time, or not logged in, try to log in */
if ($t.ProgressSession == undefined ||
    $t.ProgressSession.loginResult != progress.data.Session.LOGIN_SUCCESS) {

    /* TempTableSvc_Item_Settings is the Settings Service created when the catalog was loaded. Replace it with the Settings Service name created for your resource. */
    var settings = TempTableSvc_Item_Settings;

    /* Putting the session object into Mobile App Builder's namespace so that it can be accessed later, if needed */
    var pdsession = $t.ProgressSession;
    if (pdsession == undefined)
        pdsession = $t.ProgressSession = new progress.data.Session();

    if (settings.serviceURI == undefined || settings.serviceURI == '')
        console.log('serviceURI was not specified.' +
                    'catalogURI: ' + settings.catalogURI +
                    'resourceName: ' + settings.resourceName);

    var loginResult = pdsession.login (settings.serviceURI, "", "");
    if (loginResult != progress.data.Session.LOGIN_SUCCESS) {
        var msg = 'ERROR: Login failed with code: ' + loginResult;
        console.log(msg);
        alert(msg);
        return;
    }

    // load catalog
    pdsession.addCatalog(settings.catalogURI);
}
```

Note: The JSDO catalog is already defined through the Settings Service (see the “Updating the JSDO Service settings for the resource” section on page 122). So, the code reads the data from that. Also, if you have a
Adding a login session to access the JSDO

separate login page, you might add the code to instantiate the \texttt{Session} object to the Load event handler for that page, then add the \texttt{login()} and \texttt{addCatalog()} (for successful login) calls on the \texttt{Click} event of a login button (or similar).

5. After adding and editing the JavaScript code for the Load event, click \textbf{Add Event}, as shown in the following figure, and click \textbf{Save} to save the project.

![Adding the new Load event](image)

\textbf{Caution:} When making any change to a project, especially after adding an event handler with custom JavaScript, be sure to click \textbf{Save} immediately, or you risk losing your work.

\textbf{Login code with user authentication}

This is a more complex login code that shows how to hook up a login page (getting userid and password), and handle errors.

\textbf{Login code with user authentication} (1 of 2)

```javascript
/* if the first time, or not logged in, try to log in */
if ($t.ProgressSession == undefined ||
    $t.ProgressSession.loginResult != progress.data.Session.LOGIN_SUCCESS)
{
    /* "TempTableSvc_Item_Settings" is the Settings service created when the catalog was loaded. Replace it with the service name created for your resource. */
    var settings = TempTableSvc_Item_Settings;
    if (settings.serviceURI == undefined || settings.serviceURI == '')
        console.log("serviceURI was not specified." +
            " catalogURI: " + settings.catalogURI +
            " resourceName: " + settings.resourceName);
    /* Putting the session object into Mobile App Builder's namespace so it can be accessed later, if needed */
    var pdsession = $t.ProgressSession
    if ($t.ProgressSession == undefined)
        pdsession = $t.ProgressSession = new progress.data.Session();
```
/* replace the initial values for userName and password with the values from fields on a login page. For instance, you can use $t('userID').val() and $t('password').val() to get the values of the fields on the page, assuming 2 fields named userID and password. 
var userName = $t('userID').val();
var password = $t('password').val(); */
var userName = ""
var password = ""
try {
    var loginResult = pdsession.login (settings.serviceURI, userName, password);

    if (loginResult != progress.data.Session.LOGIN_SUCCESS) {
        var msg;
        console.log('ERROR: Login failed with code: ' + loginResult);
        switch (loginResult) {
            case progress.data.Session.LOGIN_AUTHENTICATION_FAILURE:
                msg = 'Invalid userid or password';
                break;
            case progress.data.Session.LOGIN_GENERAL_FAILURE:
                default:
                    msg = 'Service is unavailable';
                    break;
        };

        /* handle the login failure - the code depends on your application design */
        alert(msg);
        /* this would prevent a click of a button (i.e. login) from continuing, for instance */
        return false;
    }

    // load catalog
    pdsession.addCatalog(settings.catalogURI);
}
catch (e) {
    /* must have had an issue during login () */
    alert("Error logging in. 
    * serviceURI: " + settings.serviceURI + 
    "\ncatalogURI: " + settings.catalogURI + 
    "\nError: ' + e);
}
Building the sample UI

This section describes how to build a simple UI used to illustrate the data mapping features described for JSDO Services. The remaining sections then refer to the home page with a list and the detail page that allows you to perform the supported CRUD operations and refresh the list.

The home page contains a list, a search bar for filtering (with a refresh button), and a button for adding a new item, as shown in the following figure.

![Home page for the simple UI for JSDO Services](image)

You can set up the list in various ways. Using approach shown in the previous figure, you can use the built-in item label to display a given field’s value. You can also add labels (and/or a grid) to the list as shown in the following figure, if you want to add multiple fields to the list item. Why is this important? One of the fields you need to map for each list item is the record ID, so when you click the item, you go to a detail page listing the fields for that one record. The record ID identifies the record to read from JSDO local storage to display on the detail page. If you go with the first approach, and use only the built-in label of the item, you need to add JavaScript code to store the record ID for each list item. Using the approach shown in the following figure, you can add a hidden label that you can use to visually map the ID of the record.
Chapter 5: Using JSDO Services in the Mobile App Builder

Figure 14: Adding the hidden record ID to the list

**Note:** Make sure you change the **Name** property of the controls you are using for data mapping, so you can identify them.

To create a new detail page:

Click **Create New → Page**, as shown in the following figure.

Figure 15: Adding a detail page for a list item

The following figure shows the detail page to create for the simple UI:
Figure 16: Detail page for simple app with JSDO Services

It has a grid with a set of labels and input fields, and a navigation bar at the footer with three buttons. Again, change the names of the input fields to make the data mapping easier.
Steps for using JSDO Services

When you create the JSDO Services in your project, you see the following types of Services listed:

- **Instance** — After the Settings JSDO Service, this is the main JSDO Service that needs to be invoked before any of the other Services. The Instance JSDO Service instantiates the JSDO for the selected Mobile resource, making it available for use with the other JSDO Services.

- **Read** — Allows you to read data from the AppServer into JSDO local storage (invoking a Mobile read operation), and optionally read data from local storage into the UI once you have read the data from the AppServer.

- **Row** — Allows you to read a given record for a given table reference in JSDO local storage into the UI.

- **Create, Update, and Delete** — Services that by default send a request to the AppServer to add, update, and delete a record (invoking the corresponding Mobile operation). You can optionally set these Services to only operate on JSDO local storage (and not have the request go to the Mobile service). In this case, you need to handle sending the requests to the Mobile service yourself by adding custom JavaScript code to invoke the JSDO `saveChanges()` method.

- **Invoke** — Allow you to invoke custom business logic by calling specified ABL routines on the AppServer provided by the Mobile resource as methods defined directly on the JSDO (invoking a corresponding Mobile invoke operation).

To use a JSDO Service after you have configured the login session and Service settings, you follow these general steps:

1. Add the Service to an appropriate page.
2. Configure any mappings between the Service data and the UI, and add any custom code or other Services to handle Service events.
3. Invoke the Service in response to an appropriate event.

The following sections describe how to use each of the available JSDO Services:

- **Using the Instance JSDO Service**
- **Using the Read JSDO Service**
- **Using the Row JSDO Service**
- **Using the Create JSDO Service**
- **Using the Update JSDO Service**
- **Using the Delete JSDO Service**
- **Using Invoke JSDO Services**
Using the Instance JSDO Service

The first JSDO Service you need to use after the Settings Service is the Instance Service.

To use the Instance JSDO Service:

1. Click the Data tab on the home page, as shown in the following figure.

![Figure 17: Using the Data tab to add a JSDO Service to a page](image)

2. Select Service in the left-hand Add datasource combo box, then select the Instance JSDO Service (in this case, TempTableSvc_Item_JSDO) in the right-hand combo box, as shown in the following figure, then click Add.

Note: The Instance JSDO Service has “JSDO” as the last word in its identifier.
3. Change the Name for the instance of the Service added to the page. It defaults to restservice. You need to make this name unique within the application. Use a general name, like jsdoservice, if you only have one JSDO in your Mobile App, or something more specific if you are going to have more than one, like ResourceNameJSDO (in this case, ItemJSDO).

4. Invoke the Instance JSDO Service. You only need to do this once for an application. For the sample, you invoke the Service on the home page Load event. So, go back to the home page, and as shown in the following figure, on the Design tab, under Events, select:
   - home for the Component,
   - Load for the Event
   - Invoke service for the Action
   - ItemJSDO (the Instance JSDO Service added to the page) for the Service

![Figure 19: Invoking the Instance JSDO Service on a page Load event](image)

5. Click Add event to add the event to the page as shown in the following figure.
Steps for using JSDO Services

When the page loads, it executes the JavaScript added for an initial Load event handler to create the new Session object, login, and add the JSDO catalog (see the “Adding a login session to access the JSDO” section on page 125), and then invokes the Instance JSDO Service in a second Load event handler.

**Note:** The numbers next to each listed event and action are incremental values that indicate the relative order in which the event handlers are executed. If you need to change the order of actions (handlers) for a given event, you can click the arrows next to these numbers, as shown in the previous figure.

### Using the Read JSDO Service

After invoking the Instance JSDO Service, you can read the data to populate the sample list. The Read JSDO Service is the Service for both reading the data into JSDO local storage (by invoking the Mobile read operation) and populating a tabular UI, such as a list, with the records of associated table references.

First, you need to decide when you want to invoke the Service. For example, you can invoke the Read Service when the Instance Service succeeds, or on the Page show event for a page with a list. The advantage of invoking it on the Page show event is that it is also invoked when you come back from the detail page to redisplay the list with any updates made by the user. Initially, we invoke the Read Service on the Success event of the Instance Service.

**To use the Read JSDO Service:**

1. Add the service to the home page. Click the Data tab and add the Read Service, as shown in the following figure.

**Note:** Remember to change the Name of the added Read Service instance to a meaningful and unique name.
2. Add an event and action to invoke the Read Service on the Success event of the Instance JSDO Service and click Add event, as shown in the previous figure.

   **Note:** Later in this section on the Read JSDO Service is a figure that illustrates how to invoke the Read Service on the Page show event of the home page.

3. Click on Edit Mapping next to the new Read Service instance to begin setting up the data mapping for the Service. This displays tabs with parameters for both the Read Service request and response in the left panel and any local variables and UI components for the page in the right panel, as shown in the following figure. You can begin by mapping the request parameters.
Figure 22: Creating a variable to indicate if the Read Service reads local storage only

On the Request tab, you can map two request parameters: a filter string to select the records to be read, and a readLocal boolean to specify whether the JSDO should read the data from the server or local storage. During the initial load of the page, because the JSDO has no data in its local storage, or when refreshing the list from changes saved to the server, you need to read the data from the server. Otherwise, you typically want to read the data from JSDO local storage.

For the sample, we map the filter parameter to the search bar and the readLocal parameter to a local variable that you need to create.

4. To add the local storage variable, type the variable name (locReadLocal) in the Create variable field, click Create, and select Local storage variables, as shown in the previous figure.

5. You can now map the local variable to the readLocal parameter and the search bar to the filter parameter, as shown in the following figure. Just drag the parameter to the corresponding component or variable on the right-hand side.
To map the response, click the **Response** tab and drag the table reference and field from the schema in the left panel to the correct component in the right panel.

**Figure 23:** Mapping Read Service request parameters

**Figure 24:** Mapping Read Service response fields using the built-in schema labels
Note that you need to drag the table reference to the list item on the right panel (and not the list itself) in order for it to populate the entire list with a list item for each record from the table reference.

7. As mentioned previously, you need to map the record ID (_id field) to an attribute on the list item. You can map this field by adding the following custom JavaScript to the list item mapping:

```javascript
$(element).attr('recid', value._id);
```

Click Add JS next to the list item (as shown in the previous figure) and add the code as shown in the following figure.

![Figure 25: Adding JavaScript to map the record ID for the Read Service](image-url)

This sets an attribute named recid to the value from the _id field in the record. You can call it whatever you want (id, recordid, etc).

8. When you are done adding the code, click Save & Return to save the added code and return to the mapping page.

9. Note that the previous two figures show the case where you use only the built-in schema labels. So, you map the field you want to be displayed as the Text property of the list item, or provide custom JavaScript (as for _id) to create a list
item attribute to map the field. Instead, you can map each field to a corresponding custom label you have added to the list item, as shown in the following figure.

Figure 26: Mapping Read Service response fields to custom list item labels

10. When you are done mapping fields for the response, click Back to Components and save the project.

11. Before you can read the data, you need to set up the `readLocal` request parameter correctly. You can set it to `false` on the Success event of the Instance JSDO Service, because the JSDO is instantiated and initialized once during page load. Under Events, add a new action on success of the Instance JSDO Service to set the value of the `locReadLocal` variable to `false` and click Add event, as shown in the following figure.

Figure 27: Setting the `readLocal` parameter to read from the AppServer

12. If you have not done so, ensure that the new action to set the `locReadLocal` variable on the Success event of the Instance JSDO Service happens before you invoke the Read Service on that same event, as shown in the following figure.
Steps for using JSDO Services

13. Once the Read Service is invoked, you want to read from local storage from that point on. So, on the Success event of the Read Service, you can set the locReadLocal variable to true and click Add event, as shown in the following figure.

Using the Read Service with a refresh button or field value change

The sample UI used in this chapter has a refresh button next to the search bar. Typically, you want to invoke the Read Service when the refresh button is clicked (or on the Value change event of the search bar field, if you don’t have a button) to read data from the AppServer.

On the Design tab pane, under Events you can add actions on the Click event of the refresh button, or on the Value change event of the search bar field to:

1. Set the locReadLocal variable to false
2. Invoke the Read Service, setting the variable to true on success

Now, if you run the application, you can see the list populated with the records from the JSDO. If you type something in the search field and click on the refresh button, it invokes the Read Service, asking it to read the data from the AppServer again using the filter you have specified.

Note: The syntax for the filter value depends on what your AppServer is expecting.

Invoking the Read Service on a Page show event

You can invoke the Read Service on the Page show event of the page (in this case, the home page) with the list, as shown in the following figure.
An advantage of this is that the Read Service is invoked when you return to the home page from the detail page to redisplay the list with the data that is in JSDO local storage, in case the user has changed it.

Using the Row JSDO Service

The Row JSDO Service allows you to get data for a given record of a table reference in the JSDO. You can use it to populate a detail page, as shown for the sample app.

The Row Service needs the ID of the record to be returned. As shown in the mapping of the Read Service, the ID of the record is saved in the list item for each record. You also need to save the record ID in a variable for use on the detail page.

The following steps describe both the cases, whether you have saved the record ID to a hidden label of the list item or to an attribute of the list item that you create using JavaScript (see the “Using the Read JSDO Service” section on page 135). Note that you also need to set the Click event of the list item to navigate to the detail page.

To set up the Row Service to populate a detail page:

1. On the Design tab, under the Events section, you can save the record ID stored in a hidden label of the list item to a local variable, as shown below in the following figure.

![Figure 31: Setting up a list item to provide its record ID to a Row JSDO Service](image)

The Action selection (Set local storage) means that when you click on the list item, the local variable named `locItemId` is assigned the value stored in the hidden label (`item_id_Label`) of the list item used for the record ID.

**Note:** You also need to select the Bind to component check box to be able to select the control (in this case, the list item) that has the value.

If you do not use a hidden label to store the record ID and have stored it, instead, as an attribute of the list item, instead of setting the variable from the label, you need to select Run JavaScript for the Action and set the value yourself, with code like this:

```javascript
localStorage.locItemId = $(this).attr('recid');
```

The `localStorage` reference is to the location of any local variables that you create. The attribute name (in this case, `'recid'`) is whatever name you used to store the record ID when you set up the Read Service.
2. Click on **Add event**.

3. After storing the record ID for a list item in a local variable, you need to navigate to the detail page on the same **Click** event. You can add this Action for the event as shown in the following figure.

![Figure 32: Setting the Click event of a list item to navigate to a detail page](image1)

4. With the list item set up to provide record access as in the previous steps, you can display the record on the detail page using the Row Service. Go to the detail page and add the Row Service using the Data tab, as shown in the following figure.

   **Note:** Remember to give the Service a unique and meaningful name (like `itemRow`).

![Figure 33: Adding the Row JSDO Service to a page](image2)

5. Click **Edit Mapping** to set up the data mapping for the Row Service. On the **Request** tab, set the `id` parameter to the value of the `locItemId` local variable, as shown in the following figure.

![Figure 34: Mapping the record ID for a selected list item to the Row Service id parameter](image3)
6. On the Response tab, map the fields in the record returned by the Row Service to the controls on the screen, as shown in the following figure.

![Figure 35: Mapping the Row Service record to controls on the page](image)

7. After completing the data mapping, invoke the Row Service. You can invoke it on the Page show event, as shown in the following figure, so the correct record is displayed every time you navigate to the detail page.

![Figure 36: Invoking the Row Service on a Page show event](image)

8. Click Add event and save the project.

At this point, you can:

1. Run the Mobile App.

2. Click a list item to go to the detail page and see the details of the record that the list item represents.
Using the Create JSDO Service

The Create JSDO Service allows you to create a new record for a table reference in the JSDO. You can use it to create the record with initial values set either from the JSDO catalog or from the UI or custom JavaScript code. The new record can be created in JSDO local storage or also on the AppServer (the default, by invoking the Mobile create operation).

To use the Create JSDO Service, you need to:

1. Add the Service to the page where you want to invoke it.
2. Set up the request to add the record to local storage only or add it on the AppServer.
3. Map the initial values of fields in the record that you create from the UI to the corresponding UI components, or leave them unmapped to use initial values from the catalog.
4. Set the local variable that stores the record ID.
5. Invoke the Service. For example, if you have an Add button on the page where the list is located (like the + button on the sample home page), you can invoke the Service on the Click event for that button.

The following steps show how you can use this Service with the sample app.

To add a record in the sample app:

1. Add the Create Service to the home page, as shown in the following figure.

![Figure 37: Adding the Create JSDO Service to a page](image)

2. When you invoke it, the Service sends the request to the server by default. You can override this behavior by setting the localOnly parameter on the Request tab of the Create service, as shown in the following figure.

![Figure 38: Setting the localOnly parameter for a Create Service request](image)
By default, `localOnly` is `false`, which causes the Service to send the record create request to the AppServer. If you only want the record created in JSDO local storage, set that parameter to `true`. You can define a local variable that contains the value, then map the variable to the `localOnly` parameter; or you can add the following custom JavaScript to set the value:

```
return true;
```

Click **Add JS** next to the parameter and add the code as in the following figure:

**Figure 39: Setting the `localOnly` parameter for the Create Service using JavaScript**

3. Remember to click **Save & Return** to add the code you have entered.

4. The Request tab also has the schema definition for the record. So, you can map any fields to UI components or additional local variables to set them. By default, (without a field mapping) the JSDO takes the initial value for each field from the JSDO catalog (the approach taken for the sample app).

**Note:** In this case, because the Create Service is not being invoked on the detail page, you do not need to map any data fields. If you, instead, invoke the Service on the detail page, you can then map the fields from the record to the corresponding UI components on the screen.
5. Regardless of any data field mapping, you need to map the record ID to a local variable, as shown in the following figure, so you can access it later to display the newly added record.

![Figure 40: Mapping the record ID for the Create Service](image)

6. You need to invoke the Create Service, then navigate to the detail page if the Service succeeds. Because you are already on the Data tab, add the event to go to the detail page on the **Success** event of the Create Service, as shown in the following figure.

![Figure 41: Navigating to the detail page on a successful Create Service invocation](image)

7. Remember to click **Add event**.

8. Go back to the Design tab and invoke the Service by adding the **Invoke service** action on the **Click** event of the `addButton` component, as shown in the following figure.
Chapter 5: Using JSDO Services in the Mobile App Builder

Figure 42: Invoking the Create Service on a button Click event

Note that because the Create Service sets the \texttt{locItem\_Id} variable to the record ID, once the detail page is shown, it automatically displays the new record because it calls the Row Service on the \texttt{Page\_show} event to display the record identified by that variable.

Using the Update JSDO Service

The Update JSDO Service allows you to change field values in an existing record of a JSDO table reference in JSDO local storage only, or on the AppServer (the default, by invoking the Mobile update operation).

In the sample, a record can be updated from user values entered on the detail page. So, you need to add the Update Service to the detail page in order to map these values to the Service request. If you want to stay on that same page, you can then map the response back to the screen.

To update a record in the sample app:

1. Add the Update Service to the detail page as shown in the following figure.

Figure 43: Adding the Update JSDO Service to a page

2. Click \texttt{Edit Mapping} next to the Update Service to define the data mapping.

3. On the Request tab, map the UI components that provide the values to the corresponding fields, as shown in the following figure.
Figure 44: Mapping controls to fields for the Update Service request

Note that you must map the _id field so the Service can find the record you are updating. You can also map the localOnly parameter (or add JavaScript as described in the “Using the Create JSDO Service” section on page 145) to override the default behavior, which is to send the request to the AppServer.

If you plan to redisplay the record to the screen, you need to map the fields to the screen on the Response tab, as shown in the following figure.
Figure 45: Mapping fields to controls for the Update Service response

**Note:** The sample app does not redisplay the fields to the screen, but navigates to the home page, instead.

4. To navigate back to the home page, you need to add a **Navigate to page** action on the **Success** event of the Update Service under Events, as shown in the following figure.

Figure 46: Navigating back to the home page after a successful Update Service request

5. Invoke the Update Service by adding an **Invoke service** action on the **Click** event of the **buttonSave** component, as shown in the following figure.
Using the Delete JSDO Service

The Delete JSDO Service allows you to delete an existing record of a JSDO table reference in JSDO local storage only, or on the AppServer (the default, by invoking the Mobile delete operation). Two common use cases for the Delete Service include the removal of:

- A user-selected record
- A newly added record (undoing record creation)

Deleting a user-selected record

To delete a user-selected record:

1. Add the Delete Service to the detail page as shown in the following figure. You can also set an action on the Success event for the Service to navigate to the home page.

2. Click Edit Mapping next to the Delete Service.
3. On the Request tab, you must map the _id field so the Service can find the record to be deleted, as shown in the following figure.

Figure 49: Mapping the record ID for the Delete Service

Note that you can also map the `localOnly` parameter (or add JavaScript as described in the “Using the Create JSDO Service” section on page 145) to override the default behavior, which is to send the request to the AppServer.

Note: You normally do not need to map the Response tab for the Delete Service, unless you need to redisplay the record for some reason after a failure to delete the record.

4. As usual, you need to invoke the Delete Service. However, you can first ask for confirmation, perhaps with a popup window or (more simply) using some custom JavaScript that asks for confirmation, as shown in the following figure.

Figure 50: Confirming the invoke of the Delete Service

This is the code for the confirmation in the previous figure, which you run on the Click event of the Delete button:
5. **Invoke the Delete Service,** also on the Click event of the Delete button, as shown in the following figure:

![Image of invoking the Delete Service on a button Click event]

**Figure 51: Invoking the Delete Service on a button Click event**

**Note:** Ensure that the final order of actions for the Click event has the confirmation **before** the invocation of the Delete Service.

**Deleting a newly added record (undo)**

In order to undo the creation of a newly added record, you need to add some custom code to recognize that it is executing during a record Create Service. For example, on the click of the Add (+) button, you can set a local variable to identify the add mode, and on the click of the Undo (or Back button), you can invoke the Delete Service, but only if you are in the add mode. If you want to experiment with this, you can:

1. **Set a local variable** (for example, `locAddMode`) on the **Page show event** to `false`.
2. **On the Success** event of the Create Service, but before you navigate to the detail page, you can set the local variable to `true`.
3. **On the Click** event of the Undo (or Back button), you need to write JavaScript code to check the value of the local variable (perhaps ask for confirmation) and invoke the Delete Service if the variable is `true`. For example:

   ```javascript
   if (localStorage.locAddMode == "true") {
       try {
           ItemDelete.execute({});
           return; /* so that it doesn't run the next action */
       } catch (ex) {
           console.log(ex.name + ' ' + ex.message);
           hideSpinner();
       }
   }
   ```

**Note:** Change `ItemDelete` to the name you have given to the instance of the Delete Service that you added to the page.
When the code is not running in add mode (locAddMode == "false"), you might add an action to invoke the Row Service, again, so it redisplays the values from the JSDO working record on the screen to undo any editing that is in progress.

**Note:** This chapter is primarily about the data mapping for JSDO Services. You need to design any behavior you want in the application, such as enabling some buttons when appropriate (like the Save and Undo button only when in add mode, or when you change data on the screen; the Delete button only when you are not in add mode; and so on).

### Using Invoke JSDO Services

An Invoke JSDO Service allows you to invoke business logic on the AppServer that is provided by a single Mobile invoke operation supported by a JSDO. When you load the JSDO catalog for a Mobile service, the Mobile App Builder creates one Invoke Service for each invoke operation defined in the catalog. Each such Invoke Service has the same name as its corresponding invoke operation.

You can add and invoke Invoke Services in the Mobile App Builder just like any other Service and perform data mapping using the Request and Response tabs displayed when you click Edit Mapping for the Service. However, Invoke Services have pre-filled parameters only on the Request tab and nothing on the Response tab, because there is no predefined set of output parameters for a Mobile invoke operation. The catalog only defines the input parameters for the operation. This means, if you want to perform data mapping for the Service response, you must update the Invoke Service by manually adding parameters to the Response tab.

#### Updating Invoke Services with response parameters

Assuming that you have created a Mobile service with an invoke operation that returns the data you want to work with in the JSDO, after you load the catalog, you see the Invoke Service that corresponds to this Mobile operation on the list of Services, as shown in the following figure for a Mobile invoke operation named `CustomReadTempTable`. 

---

*OpenEdge® Development: Mobile Applications*
Steps for using JSDO Services

Figure 52: Updating an Invoke Service definition—the Request tab

The previous figure, where you have the Invoke Service selected, also shows the Request tab in the right panel, which you do not typically change, unless you want to define a default value for one of the pre-defined parameters.

The first thing you need to do, then, if you want to map data from the Invoke Service response, is to go to the Service definition and update the Response tab.

On the Response tab, you can enter the name of each output parameter and click Add to add the parameters for the response, as shown in the following figure for the `myOutParam` and `ttItem` parameters.
Chapter 5: Using JSDO Services in the Mobile App Builder

Figure 53: Updating an Invoke Service Response tab with a temp-table

Notes: The parameters you add to the Response tab are the output parameters of the corresponding ABL routine on the AppServer. Using default mapping for the Mobile invoke operation, these ABL output parameters are returned as corresponding properties and objects with the same names as the output parameters. These “output” properties and objects are contained in a response object referenced by the response property of the request object that is returned by the invocation method on the JSDO. This response object, then, contains a property for each scalar output parameter and an object for each temp-table or ProDataSet output parameter.

Note also that regardless of the output parameters returned, you only need to add parameters to the Response tab that you must access in your application. For example, in the previous figure, you can omit myOutParam if you only need to access ttItem.

As shown in the preceding figure, you type the name of the property or object returned in the fill-in field next to the Add button, and click Add. Note that for temp-tables, the response contains the name of the serialized temp-table as the parameter, which in turn contains the name of the temp-table as an array. So, you have to add a level with the name of the temp-table and set it as an array. This is why, in the previous figure, the temp-table name, ttItem, appears twice. Finally, you add a third level for the fields returned in the temp-table. If you are returning a simple scalar parameter, you specify the parameter only at the top level (myOutParam in the previous figure).

Note that if you are returning a ProDataSet parameter, the ProDataSet is the name that you repeat for two levels in the Response tab, and add a third level for the temp-table array, for a total of four levels as shown in the following figure for the dsItem parameter.
Steps for using JSDO Services

OpenEdge® Development: Mobile Applications 157

Figure 54: Updating an Invoke Service Response tab with a ProDataSet

Note: Any scalar array parameter or temp-table field, you also specify as an array object with the parameter or field name listed twice, at both the array and field level.

Once you make these changes (if necessary), you can add and invoke the Service on a page, map request and response parameters as you require, and process the response similar to any JSDO Service.

Following are some sample use cases for the Invoke JSDO Service.

Invoking a custom read operation

You might want to write your own version of the JSDO fill( ) method that supports the Mobile read operation (for example, taking more input parameters). You have to do this through a Mobile invoke operation because the signature for the Mobile read operation is prescribed and cannot be changed. After returning the JSDO data with your custom read method, you can use the JSDO addRecords( ) method to add the records to JSDO local storage (necessary if you want to use the Create, Update, or Delete Services, or any other functionality provided by the JSDO).

To invoke a custom read operation using an Invoke Service:

1. Add the Invoke Service to the page and invoke it where appropriate. For example, using the Read Service sample described previously (see the “Using the Read JSDO Service” section on page 135), you can invoke the Service on the Page show event.

2. On the Data tab, click Edit Mapping next to the Invoke Service and map the parameters on the Request tab like any other Service.

3. Because you need to add the data to JSDO local storage, do not perform data mapping for the Response tab. Instead, use custom JavaScript to add the data.
Add a Run JavaScript action on the Success event of the Invoke Service and write the code, as shown in the following figure.

```javascript
ROWSER
/* need to put the data into the JSDO’s local storage */
var jsdo = itemJSDO.service.jsdo;

/* add records to the local local cache, where: 
- data is an object passed to the on success function with the 
  results from the invoke operation. The temp-table will be in an 
  object with the same name, so that’s why you need to pass data.ttItem 
- progress.data.JSDO.MODE_EMPTY so that the JSDO empties the local 
  cache */

jsdo.addRecords(data.ttItem, progress.data.JSDO.MODE_EMPTY);
```

Figure 55: Adding data records from an Invoke Service to JSDO local storage

This is the code shown in the previous figure:

```javascript
/* need to put the data into the JSDO's local storage */
var jsdo = itemJSDO.service.jsdo;

/* add records to the local local cache, where:
- data is an object passed to the on success function with the
  results from the invoke operation. The temp-table will be in an
  object with the same name, so that's why you need to pass data.ttItem
- progress.data.JSDO.MODE_EMPTY so that the JSDO empties the local
  cache */

jsdo.addRecords(data.ttItem, progress.data.JSDO.MODE_EMPTY);
```

In this sample code, itemJSDO is the name of the Invoke Service instance you have added to the page, which allows you to return a reference to the JSDO. The data from the Service is in the data object, which is passed to the JavaScript code. You can then call the JSDO addRecords() method passing the object with the temp-table or ProDataSet to be added to the JSDO local storage.

Mapping the response for an Invoke Service

This is a case for mapping the response for an Invoke Service. In this sample, an ABL class method that implements the Mobile invoke operation has a return value and an output parameter that you can map to a counter value defined for a list item.

To map the response for this Invoke Service:

1. Update the Response tab for the Invoke Service definition to add the return value and output parameter, as described previously (see the “Updating Invoke Services with response parameters” section on page 154), and as shown in the following figure.
Figure 56: Adding an ABL return value and output parameter to the Invoke Service Response tab

**Note:** The \_retVal parameter is an OpenEdge-defined property in the response object that holds the return value for the non-void ABL method, GetNumBins(), that implements the Mobile invoke operation.

2. Add the Invoke Service to the page, and invoke the Service.

3. Open the Response tab mapping as you normally do, and map the response similar to the following figure:

![Mapping an output parameter on an Invoke Service Response tab](image)

Figure 57: Mapping an output parameter on an Invoke Service Response tab

In this sample, both \_retVal and pOutNumBins return the same value. So, you can map either one and completely omit the one from the Service definition that you do not use.
The following figure shows the detail page at run time, with the list counter value populated using data from this Invoke Service.

![Sample page showing Invoke Service parameter mapping](image)

**Figure 58: Sample page showing Invoke Service parameter mapping**

You can also add code to disable a list item based on a response value.

**To disable a list item if the value of the response parameter is less than or equal to zero:**

1. In the Response tab of the mapping editor, click **Add JS** next to the **Counter value** element, shown mapped to `_retVal` in the following figure.

![Mapping the Invoke Service return value parameter](image)

**Figure 59: Mapping the Invoke Service return value parameter**
2. Add code similar to the following figure.

```javascript
function(value, element) {
    if (value <= 0) {
        $(element).parent().addClass('ui-disabled');
    } else {
        $(element).parent().removeClass('ui-disabled');
    }
}
```

**Note:** Remember to click Save & Return to exit out of the JavaScript editor.

This is the code in the previous figure:

```javascript
if (value <= 0)
    $(element).parent().addClass('ui-disabled');
else
    $(element).parent().removeClass('ui-disabled');
```

Because you have added the code to the Counter value element, and you want to disable the list item (parent of the Counter value element), the code is calling the appropriate method on the parent (parent()) of the current element ($element).

The following figure shows the detail page as it looks when the counter value is zero:
Figure 61: Sample page showing Invoke Service parameter test result
Error handling

Mobile App Builder Services provide a means to handle errors. For example, if the Read JSDO Service fails, you do not get any data to populate the list, but you probably want to perform some task, such as going to a page or displaying a pop-up (for example) that alerts the user that the data cannot be returned.

Mobile App Builder Services have an Error event that you can use to trap errors during the execution of a Service. On the Data tab, just as you have a Success event, you can also select the Error event, as shown in the following figure.

Figure 62: Error handling for JSDO Services

You can also select Run JavaScript as the action for the event and write some code to assign the error information to a local variable so you can display it on a pop-up, in a message, etc. For example, this is some simple code to get information for the error:

```javascript
/* input parameters are: jqXHR, textStatus */
var msg = "";
if (textStatus != undefined)
    msg = textStatus;
if (jqXHR)
    msg = msg + " Status :" + jqXHR.status;
alert("ERROR : " + msg);
```
Chapter 5: Using JSDO Services in the Mobile App Builder

Note that you have access to the XMLHttpRequest (XHR) object (when available) so you can try to get more information, if needed.

**Error handling for the Update Service**

The Update Service has a caveat on an error condition, if you have added the record to JSDO local storage only (that is, you set `localOnly` to `true` on the Create Service) and the record has not been sent to the AppServer yet. When you invoke the Update Service with the default behavior, the record is sent to the AppServer as a Mobile create operation. The default behavior of the JSDO is to undo the operation on error, so the record will be deleted from local storage. The Update Service compensates for that by reading the record that failed to be created on the AppServer back into the JSDO local storage. However, the record ID is now different from the previous version. So, if you have saved the record ID to a local variable, you have to reset this variable so the Update Service can find it when you retry the operation.

This is sample code that you can add to the `Error` event of the Update Service, replacing `ttItem` with the name of your temp-table:

```javascript
if (jqXHR.request && jqXHR.request.operation == progress.data.JSDO._OP_CREATE) {
  /* should have current record if it failed on a create */
  if (jqXHR.jsdo.ttItem.record != undefined) {
    /* the id will be different, so refresh it */
    if (jqXHR.jsdo.ttItem.record.getId() != localStorage.locItemId) {
      localStorage.locItemId = jqXHR.jsdo.ttItem.record.getId();
    }
  }
}
```

**Error handling for the Read Service**

The JSDO’s default behavior is to empty its local storage before processing the response from a Mobile read operation, even if the response returned is an error condition. If you have mapped the response of a default Read Service to a list (with `readLocal` set to false), the list is not automatically refreshed to display no records on an error condition. You must handle this error situation: clear the list, go to a different page, log the user out, etc.

If you want to clear out the list, you can use two approaches. On the `Error` event of the Read Service, you can:

- Invoke the Read Service, again, setting `readLocal` to `true`. Because JSDO local storage has no records, this causes the list to be refreshed with no records.
- Write JavaScript like this to clear the list yourself, replacing `CustList` with the name of the list on your page:

```javascript
/* save template (first li) before removing all items from list */
var tmpl = $t('CustList').children().eq(0);
$t('CustList').html('').append(tmpl);
$t('CustList').listview('refresh');
```
Deploying Mobile Applications

This chapter provides information on production deployment for both Mobile Web applications and the Mobile Apps that access them. Production deployment considerations can also be helpful for Mobile application developers who need to test their Mobile Apps and services prior to deployment. Also, the Mobile App developer needs to know some production security considerations in order to properly design and code their Mobile App.

The following sections describe these deployment considerations in detail:

- Deployment overview
- Packaging and deploying Mobile services
- Packaging and Deploying iOS Apps
- Packaging and Deploying Android Apps
- Packaging and Deploying Mobile Web Apps
- Security considerations
Deployment overview

There are three options for packaging and deployment of a Mobile App:

- iOS App
- Android App
- Mobile Web App

Table 3 lists the terminology used in this chapter to describe the considerations for deployment of different types of mobile applications.

Table 3: Mobile Application Terminology

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mobile Web application</td>
<td>The Mobile App and/or Mobile services deployed to a web server</td>
</tr>
<tr>
<td>Mobile services</td>
<td>The services deployed to a web server and accessed by a Mobile Native App or Mobile Web App</td>
</tr>
<tr>
<td>Mobile App</td>
<td>Any Mobile client UI that runs natively on a mobile device (Mobile Native App) or in a web browser on a mobile device (Mobile Web App)</td>
</tr>
<tr>
<td>Mobile Native App</td>
<td>An application that runs natively on a mobile device</td>
</tr>
<tr>
<td>Mobile Web App</td>
<td>An application that runs in a web browser on a mobile device</td>
</tr>
<tr>
<td>Mobile application</td>
<td>A Mobile App and all of its associated backend services</td>
</tr>
</tbody>
</table>
Packaging and deploying Mobile services

To package Mobile services for deployment (with or without a Mobile Web App included), you can use the Export as Mobile Web Application wizard in Progress Developer Studio for OpenEdge (Developer Studio).

To package Mobile services:

1. In Developer Studio, right-click on the name of your project in the Project Explorer view and choose Export → Mobile Web Application.

2. Choose the Mobile Project name, Destination, and Mobile/REST services. Optionally, choose a Mobile Web App to package with the services.

3. Checking the Include default runtime library files box includes the files necessary to create a standalone Mobile Web application that can be deployed on any server.

4. Click Finish. The WAR file is created and placed in the destination you selected.

Deployment can be done by using OpenEdge Explorer or OpenEdge Management to deploy the WAR file to an OE Web Server. This applies the default settings necessary to enable and make the Mobile Web application available for access by Mobile Apps. Alternatively, you can manually copy the WAR file to the webapps folder of an Apache Tomcat server, which requires manual configuration of the settings necessary to make the Mobile Web application available for access by Mobile Apps.

The tools and options for deploying the WAR file for a Mobile Web application are similar to those for deploying the WAR file for a REST Web application. For more information, see the sections on REST administration in OpenEdge Application Server: Administration.
Packaging and Deploying iOS Apps

Packaging a Mobile App for iOS (IPA file) requires a certificate and a provisioning profile. These credentials are required during development as well as deployment. For more information on creating the certificate and provisioning profile, see the “Deploying iOS Apps” section on page 170.

Note: When deploying iOS Apps to different environments (development and testing vs. production deployment), different source code must be deployed. The URIs for logging into Mobile Web applications and accessing their services in each environment must be different absolute URIs. The JavaScript source for production deployment should have the required absolute URIs hard coded to log into all Mobile Web applications whose Mobile services are accessed by the iOS App.

Packaging iOS Apps

To build and package a Mobile App for iOS:

1. In Progress Developer Studio for OpenEdge (Developer Studio), right-click on the project and choose Properties → Progress OpenEdge → Mobile App Builder.
2. On the Build options page, select iOS. Click OK.
3. Expand the Mobile Apps folder in the Project pane and double-click on the name of the Mobile App. The Mobile App Builder will open in a browser.
4. In the Mobile App Builder, expand Project and click App settings. Click the iOS binary IPA properties tab.
5. Enter the Mobile App name in the Label field.
6. Enter the Bundle ID in the Bundle ID field.
7. The icon is the graphic that will launch the Mobile App on an iOS device. Upload an icon file using the Browse button in the Icons area.
8. Upload a launch image file for the application using the Browse button in the Launch Images area.
9. To upload the certificate:
   a. Click Change next to Certificate file.
   b. Click Upload file.
   c. Click Upload a file and locate the certificate. Once the upload is complete, click Back to files list.
   d. Click on the certificate file and click Select.

10. Enter the certificate password.

11. To upload the provisioning profile:
   a. Click Change next to Provisioning profile file.
   b. Click Upload file.
   c. Click Upload a file and locate the provisioning profile. Once the upload is complete, click Back to files list.
   d. Click on the provisioning profile file and click Select.

12. Click Save.

13. In Developer Studio, right-click on the name of the Mobile App in the Mobile Apps folder and choose Copy source local to get the updated Mobile App in the Developer Studio project.

14. To package the IPA, right-click on the name of the Mobile App in the Mobile Apps folder and choose Export local. The IPA file will in the bin directory.

You can test your app during development by copying the IPA directly to an Apple device.

To copy the IPA to an iOS device:
1. Open iTunes on your computer.
2. Drag the IPA into the iTunes Library.
3. Plug the iOS device into the computer.
4. Drag the IPA from the iTunes Library onto the icon showing the device.

If you get an installation error, verify that the UUID of the device is registered correctly and that the provisioning profile includes that device. For more information, see the “Deploying iOS Apps” section on page 170.
Deploying iOS Apps

Building a Mobile App for iOS (IPA file) requires registration as an Apple Developer, a developer certificate, and a provisioning profile. The following sections reference information to help you create your own credentials.

**Note:** Though not required, you can do this more easily on a Mac.

For more detailed information visit:


**Apple Developer**

To register as an Apple Developer, visit this website:


**iOS Provisioning Portal**

To create the certificate and provisioning profile, use the iOS Provisioning Portal. This portal is part of the iOS Dev Center. To access the portal, log in at the following link with the Apple ID you used to register as a developer:


Click on the iOS Provisioning Portal link.

**Certificates**

The following video has more information on certificates:


Note that a certificate must be in P12 format in order to be used in the Mobile Application Builder.

**App IDs**

An App ID has three parts: Description, Bundle Seed ID (App ID Prefix), and Bundle Identifier (App ID Suffix). The Description of the App ID can be anything, e.g., OEMobile. The Bundle Seed ID (App ID Prefix) is assigned for you and is assigned to be your Team ID. The Bundle Identifier can be unique for each application or use a wildcard for many applications. Although it is possible to use `*` by itself, the recommended practice is to use the following notation: `com.progresssoftware.*` or `com.progresssoftware.applicationsname`.

It is important to remember what you entered for the Bundle Identifier. This will be used when creating the IPA in the Mobile App Builder. The following link describes Bundle Identifiers:


The following video has information about app IDs:

Devices

Each device must be registered. This is accomplished by using the device UUID. The UUID can be found by running a free UUID app on the device.

The following video has more information on devices:

https://developer.apple.com/ios/manage/overview/index.action#

Provisioning Profiles

A Provisioning Profile has four parts: Profile Name, Certificate(s), App ID, and Device(s). The Profile Name can be anything, e.g., OEMobile. The following video has more information on provisioning:

https://developer.apple.com/ios/manage/overview/index.action#

The following link also has more information:

Packaging and Deploying Android Apps

Packaging a Mobile App (APK file) for Android requires a signing key/certificate, but during development and testing, the signing key/certificate inputs are not required. For more information on creating the signing key/certificate, see the “Deploying Android Apps” section on page 174.

**Note:** When deploying Android Apps to different environments (development and testing vs. production deployment), different source code must be deployed. The URIs for logging into Mobile Web applications and accessing their services in each environment must be different absolute URIs. The JavaScript source for production deployment should have the required absolute URIs hard coded to log into all Mobile Web applications whose Mobile services are accessed by the Android App.

-Packaging Android Apps

To build and package a Mobile App for Android:

1. In Progress Developer Studio for OpenEdge (Developer Studio), right-click on the project and choose Properties → Progress OpenEdge → Mobile App Builder.
2. On the Build options page, select Android. Click OK.
3. To open the Mobile App within the Mobile App Builder, double-click on the name of the Mobile App. The Mobile App Builder will open in a browser.
4. From within the Mobile App Builder, choose Project → App settings and click Android binary APK properties.

![Android App Builder Screen](image)

5. Enter the Mobile App name in the Label field.
6. The launcher icon is the visual representation of your app on the mobile device. To upload an icon:
   a. Click Change next to Icon.
   b. Click Upload file.
   c. Click Upload a file and locate the icon file. Once the upload is complete, click Back to images list.
   d. Click on the icon file and click Select.
   e. Click Upload a file and locate the provisioning profile. Once the upload is complete, click Back to files list.
   f. Click on the provisioning profile file and click Select.

7. Enter the Version code.

8. Enter the Version name.

9. Enter the Package name.

10. Enter the Minimum SDK version.

11. Enter the Target SDK version.

12. Enter the Maximum SDK version.

13. To upload the Keystore file:
   a. Click Change in the Keystore file field. Click Upload file.
   b. Click Upload a file and locate the Keystore. Once the upload is complete, click Back to files list.
   c. Click on the Keystore file and click Select.
   d. Enter the Key alias.
   e. Enter the Key password.
   f. Enter the Keystore password.

14. Click Save.

15. In Developer Studio, right-click on the Mobile App in the Mobile Apps folder and choose Copy source local to get the updated Mobile App to the Developer Studio project.

16. To package the APK, right-click on the Mobile App in the Mobile Apps folder and choose Export local. The APK file will in the bin directory.
During development, there are several ways to get the newly created APK on an Android device for testing. One

2. Click and drag the APK into dropbox.
3. Open http://dropbox.com from a browser on the mobile device and log in.
4. Click on the APK from within the dropbox application. This will begin downloading the application onto the device.
5. When the download is complete, click on the downloaded file on the mobile device. Follow the device directions for installing the app.

Deploying Android Apps

For detailed information on Android development and deployment, visit this website:


Versioning your Mobile App

Versioning is important for application upgrades and maintenance. Users need to have specific information about the application version that is installed on their devices and the upgrade versions available for installation, and other applications and services may need to check the version of your application for compatibility issues. For more information go to:


Package Name

The Application package is an identifier of the application. This identifier is unique among all the apps installed on device at a given moment; there cannot be two apps with the same Application package installed at the same time. It is also unique on the Android market; there cannot be two apps with the same Application package on the Market. Conflict over the Application package with unforeseen third-party apps is possible. Using the Java package name convention ("com.mydomain.myapp") for the Application package name is recommended to avoid such conflict.

Signing your application

The Android system requires all installed applications be digitally signed with a certificate whose private key is held by the application's developer. To test and debug your application, the build tools sign your application with a special debug key. When you are ready to release your application for end-users you must sign it with a suitable private key. For more information go to:

Packaging and Deploying Mobile Web Apps

You can package a Mobile Web App with or without associated Mobile services using Developer Studio.

**Note:** When you deploy a Mobile Web App to the same Web server as the Mobile Web application that it accesses, its URIs for logging into the Mobile Web application and accessing its services can be relative to the host location (or domain) of the Web server where both the Mobile Web App and Mobile Web application are deployed. However, when a Mobile Web App and its Mobile Web application are deployed to different Web servers, the JavaScript for the Mobile Web App must be hard coded with the absolute URIs to access the Mobile Web application and its services. This also means that when deployed to different environments (development and testing vs. production deployment), you must deploy different source code to allow the Mobile Web App to access the Mobile Web application and its services in each environment.

Packaging Mobile Web Apps

To package a Mobile Web App with associated Mobile services, you can use the Export as Mobile Web Application wizard in Developer Studio. A single Mobile Web App can be packaged with the Mobile service.

**To package a Mobile Web application with mobile services:**

1. In Developer Studio, right-click on the name of your project in the Project Explorer view and choose **Export → Mobile Web Application**.
2. Choose the Mobile Project name, Destination, Mobile App, and Mobile/REST services.
3. Checking the **Include default runtime library files** box includes the files necessary to create a standalone Mobile Web application that can be deployed on any server.
4. Click **Finish**. The WAR file is created and placed in the destination you selected.

**To package a Mobile Web application without mobile services:**

Right-click on the application name in the Project Explorer view in Developer Studio and choose **Export Local**. The packaged app will appear as a ZIP file in the bin directory. (The default export format is WebApp, but you can change that by navigating to **Properties → Progress OpenEdge → Mobile App Builder** and selecting a native format.)
Deploying Mobile Web Apps

Deployment can be done by using OpenEdge Explorer or OpenEdge Management to deploy the WAR file to an OE Web Server. This applies the default settings necessary to enable and make the Mobile Web application available for access by Mobile Apps. Alternatively, you can manually copy the WAR file to the `webapps` folder of an Apache Tomcat server, which requires manual configuration of the settings necessary to make the Mobile Web application available for access by Mobile Apps.

The tools and options for deploying the WAR file for a Mobile Web application are similar to those for deploying the WAR file for a REST Web application. For more information, see the sections on REST administration in *OpenEdge Application Server: Administration*.

For more information on packaging a Mobile App as part of a Mobile Web application, see the new “Packaging and deploying Mobile services” section on page 167.
Security considerations

The security considerations for OpenEdge Mobile applications described in this section apply variously to both development environments (which might be managed by a developer or development administrator) and production environments (which are often managed by an IT administrator). For specific considerations that apply to the development and testing of Mobile services, you can also find more information in Chapter 3, “Creating Mobile Services.” For specific considerations that apply to the development and testing of Mobile Apps, you can also find more information in Chapter 4, “Creating Mobile Apps using JSDOs.” Generally, this section addresses the security requirements of production environments unless otherwise indicated. Developers will also be interested in this information when they need to make application design and coding decisions for a particular production environment.

In general, security considerations for OpenEdge Mobile applications using the HTTP/HTTPS REST transport include the following:

- **Web server authentication models** — The Mobile Web application can be configured to support one of several Web server authentication models in order to authenticate user access to its Mobile App and Mobile services. OpenEdge Mobile supports a subset of the available Web server authentication models, and many of a Web application’s security options depend on the authentication model that you select.

- **SSL connections** — OpenEdge allows you to configure SSL connections, which enables your Mobile application to use URIs based on HTTPS between the Mobile App and its Mobile Web application, and AppServer SSL between the Mobile Web application and the AppServer. Each SSL connection enables secure data exchange between a Mobile App and its Mobile resources running in an AppServer.

- **Mobile device security** — You might need to consider options for securely hosting a Mobile App on a particular mobile device, such as securing the Mobile App on the device or registering a public certificate for the Web server where Mobile services are deployed to allow SSL connections to these services.

The following sections describe how these security considerations affect Mobile application deployment.

### Web server authentication models

The Mobile Web application must be configured to use one of the authentication models supported by OpenEdge Mobile.
OpenEdge Mobile supports the following Web server authentication models:

- **Anonymous** — The Web server requires no user authentication or login session management for access.

- **HTTP Basic Authentication** — The Mobile App must obtain and send the required user name and password to the Mobile Web application in order to access the requested resource. Note that the Mobile App (using a `Session` object) sends the required user name and password automatically (assuming that authentication has not already occurred) if you provide them at session login.

- **HTTP Form Authentication** — If the Mobile App requests a resource that requires authentication it may take one of two actions: 1) respond to an HTML user login page from the Mobile Web application after it references the resource, or 2) send the Mobile Web application the required user ID and password before requesting the resource. If the Mobile Web application can validate the user with the credentials provided, it allows the client access to the resource.

**Note:** Progress supplies application security templates that are used to define the protection on the Mobile services and/or the Mobile App (when the Mobile App is a Mobile Web App deployed to the Web server). These templates are files named `appSecurity-authentication_type.xml`, where `authentication_type` indicates the Web server authentication model. These files are located in the `WEB-INF` directory of an OpenEdge Mobile project in Progress Developer Studio for OpenEdge. See the templates for information on specific protection schemes, and see the sections on configuring security for REST applications in *OpenEdge Application Server: Administration* for additional information on Mobile Web application security.

Your Mobile App design can significantly affect how it responds to the selected Web server authentication model. For more information, see the sections on creating a user login session in Chapter 4, “Creating Mobile Apps using JSDOs.”

To operate effectively together, a Mobile Web application and any Mobile App that accesses its Mobile services must be configured and coded (respectively) by the developer for the same Web server authentication model.

To configure the Web server authentication model for a Mobile Web application, you use the same options for choosing the security model for a REST Web application. These options work together with the Spring Security framework installed with OpenEdge to manage security for REST applications. For more information, see the sections on choosing and applying a security configuration for REST applications in *OpenEdge Application Server: Administration*.

**Note:** If you select HTTP Form Authentication, the mobile devices and Web browsers that access the Mobile Web application **must** have cookies enabled. Otherwise, the Mobile App cannot login and access Mobile services.

If you select HTTP Basic Authentication and the mobile devices and Web browsers do not have cookies enabled, you **must** set a property in the SSO configuration of the Mobile Web application to allow session logins from Mobile Apps. For more information, see the “Enabling SSO for a Web application” section on page 179.
In order to avoid certain problems that can result when authenticating from different mobile devices and Web browsers, you must ensure that any Mobile App that accesses the services of a Mobile Web application is built to use the same Web server authentication model as the Web application it is accessing. The developer must code the Mobile App to use the appropriate authentication model. For more information on coding the authentication model for a Mobile App, see the sections on creating a user login session in Chapter 4, “Creating Mobile Apps using JSDOs.”

The following is a series of general steps you can follow to configure the security features associated with your selected authentication model:

To configure the Mobile Web application security features associated with its selected Web server authentication model:

1. If you have coded the ABL application services on the AppServer to rely on a single sign-on (SSO) client-principal passed with each request from the Mobile Web application to establish an AppServer login session for each request, you must enable the Web application for SSO.

2. For the selected authentication model, you can add application users and their access roles for the Mobile Web application.

3. Once you have configured user access roles, you can define access control lists for the resources requested by a Mobile App based on these access roles.

4. You can also configure the Mobile Web application to support cross-origin resource sharing (CORS), which allows a Mobile App to make requests for Mobile resources hosted in different Web domains.

As with configuring the Web server authentication model, you can configure associated security features for a Web application using the options available to configure these same features for a REST application. The following sections provide more information on configuring each of these features.

Enabling SSO for a Web application

If enabled, your Web application will create an SSO client-principal for the user credentials authenticated by the Mobile Web application. When the Mobile App executes an operation for a supported Mobile resource, the Mobile Web application passes its authenticated user ID and login session ID in the form of a client-principal to the AppServer. The AppServer can use it to establish the login session to complete the operation request. For more information on establishing an SSO login session on the AppServer, see the sections on AppServer coding in Chapter 3, “Creating Mobile Services.” For information on enabling SSO for an OpenEdge REST or Mobile Web application, see the sections on SSO support for REST applications in OpenEdge Application Server: Administration.

Note: If you have selected HTTP Basic Authentication for the Web application and cookies will not be enabled for the mobile devices and Web browsers that access its Mobile services, you must set the ccid property appropriately in your SSO configuration in order to allow the session ID to be passed between the Mobile App and its Web application.
Defining user roles for a Web application

User roles allow you to define access controls for the Mobile services and resources of a Mobile Web application so that only users that have a given assigned role can access a given resource. OpenEdge provides a built-in set of predefined roles or you can define your own user roles for a Web application. For more information, see the sections on adding users, and user roles and privileges, for REST applications in OpenEdge Application Server: Administration.

Setting access controls based on user roles

The tools for setting access controls on a Web application's resources are provided entirely using the Spring Security framework installed with OpenEdge. For more information on using the Spring Security framework to apply access controls, see the Spring Security documentation at http://static.springsource.org/spring-security/site/reference.html.

Cross-origin resource sharing (CORS)

Cross-origin resource sharing (CORS) is a W3C group standard that allows a Mobile App's JavaScript to access Web application resources in a DNS domain different from the one the current HTTP page and JavaScript were loaded from. Such "cross-domain" requests are otherwise forbidden by Web browser's JavaScript engine. The CORS standard defines a way in which a Mobile App's JavaScript can ask the Web application if it can make the cross-origin request, and the Web application's configuration can determine if the cross-domain request will be granted. The W3C CORS standard works by adding new HTTP headers that allow servers to control resource access to permitted origin domains.

CORS support is enabled in Mobile Web applications and its defaults are configured to grant access to all Mobile services from any generic HTTP requests (made by a non JavaScript client) and any JavaScript engine from any DNS domain. This means that a Mobile App can load a Web page from one DNS domain and perform Mobile operations on any Mobile service's resources residing in another DNS domain. CORS support is extended to all modern mobile devices and Web browsers. Before using devices and browsers with a CORS-enabled Mobile application, ensure that they support the CORS standard. For more information on the CORS standard, see the documentation at http://www.w3.org/TR/cors/.

You might need to configure CORS support for a Mobile Web application specific to a production site's requirements. You can do this in exactly the same way as for a REST Web application. OpenEdge supports CORS configuration using the Spring Security framework embedded in each Mobile and REST Web application. For more information on configuring this CORS support, see the sections on managing security for REST applications in OpenEdge Application Server: Administration.
SSL connections

Secure Sockets Layer (SSL) allows you to configure secure (encrypted) HTTPS connections between your Mobile App and the Mobile services hosted in your Mobile Web application. You typically configure a Mobile Web application to use HTTPS connections when the data being sent between the Mobile App and the Web application is restricted to only authenticated users or when logging into the Mobile application (to protect the exchange of user credentials).

When a Mobile application is configured to use HTTPS, the Mobile App communicates with its Mobile services using URIs based on HTTPS (HTTP over SSL). To enable SSL, you configure appropriate digital certificate stores for the Web server and configure the Mobile Web application's security policy to require HTTPS connections from its clients (Mobile Apps). You might also require the use of SSL connections between the Mobile Web application and its ABL application services on the AppServer when the AppServer is located on the site’s intranet. You also need to ensure that the mobile devices or Web browsers that run the Mobile App are configured to validate the Certificate Authority (CA) public certificates who issued the Web server's server certificate.

You can configure HTTPS for a Mobile application in exactly the same way as for a REST application. For more information on configuring a REST application and its AppServer for SSL, see the sections on SSL support for REST applications in OpenEdge Application Server: Administration. For more information on using SSL in OpenEdge generally, see OpenEdge Getting Started: Core Business Services - Security and Auditing.

Mobile device security

The options for mobile device security, such as securing Mobile Apps on a mobile device or support for public certificates to allow SSL access to Mobile services depend on the particular device or Web browser. For more information, see the documentation on hosting Apps for the particular device or browser.
ABL to JavaScript Data Type Mapping

An OpenEdge progress.data.JSDO (JSDO) is a JavaScript class instance that provides the Mobile App behavior for a data resource that is implemented in ABL. A JSDO communicates with the OpenEdge AppServer through a Mobile service to marshal ABL data for parameters (and return values) of ABL routines that pass the following ABL data types: ProDataSets, temp-tables, arrays, and primitive values. The JSDO provides this ABL data to the Mobile App programmer as JavaScript data types. For more information on ABL data types, see the reference entry on data types in OpenEdge Development: ABL Reference. For more information on the format and use of JavaScript data types, you can review sources on the Web, such as http://www.w3schools.com/js/js_datatypes.asp.

The following sections present a brief description of JavaScript data types and how they map to data types in ABL:

- JavaScript data type overview
- Data type mapping between JavaScript and ABL

For more information on JSDOs and how they communicate with an AppServer through a Mobile service, see the "progress.data.JSDO class" section on page 191.
JavaScript data type overview

JavaScript supports four primitive data types, as shown in Table 4.

<table>
<thead>
<tr>
<th>Data type</th>
<th>Description</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>A string of characters enclosed in double or single quotes.</td>
<td>&quot;jump rope&quot;</td>
</tr>
<tr>
<td>Number</td>
<td>An unquoted numeric value, which can include an exponent using scientific notation.</td>
<td>17, 54.35, 0.9582e-42</td>
</tr>
<tr>
<td>Boolean</td>
<td>The unquoted, lowercase, literal value true or false.</td>
<td>true</td>
</tr>
<tr>
<td>null</td>
<td>The unquoted, lowercase, literal value, null.</td>
<td>null</td>
</tr>
</tbody>
</table>

Table 4: JavaScript primitive data types

The data type of a primitive value is determined by the format of the value:

- A string of characters surrounded by quotes indicates the value is a String.
- A string of numeric characters—without quotes—with an optional decimal point, an optional negative sign, or an optional exponent indicates a Number.
- The string true or false—without quotes—indicates a Boolean.
- The unquoted string null indicates a null (or undefined) value, which maps to the ABL Unknown value (?).

In addition to these standard primitive data types, there are some commonly-used (though not officially supported) data types for certain values. For this purpose, specially formatted strings represent values for which there is no standard primitive JavaScript data type.

Table 5 shows the non-standard data types that ABL supports.

Table 5: Supported non-standard data types

<table>
<thead>
<tr>
<th>Non-standard data type</th>
<th>Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date</td>
<td>A String in the ISO 8601 format, &quot;yyyy-mm-ddThh:mm:ss.sss+hh:mm&quot;. JavaScript does support a Date object for working with dates and times. However, all dates and times returned from the AppServer to an OpenEdge JSDO are stored as a String in the ISO 8601 format.</td>
</tr>
<tr>
<td>Binary data</td>
<td>A String consisting of the Base64 encoded equivalent of the binary data</td>
</tr>
</tbody>
</table>
JavaScript also supports two complex data types, used to aggregate values of all JavaScript data types, including both primitive and complex data, as shown in Table 6:

Table 6: JavaScript complex data types

<table>
<thead>
<tr>
<th>Data type</th>
<th>Description</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object</td>
<td>A comma-delimited list of named values (properties), either primitive or complex, enclosed in braces ({}). The property names can either be literal values or quoted strings.</td>
<td>{ myString : &quot;jump rope&quot;, 'myNum' : 17, 'myBool' : false }</td>
</tr>
<tr>
<td>Array</td>
<td>A comma-delimited list of unnamed values, either primitive or complex, enclosed in brackets ([[]])</td>
<td>[ &quot;jump rope&quot;, 17, false ]</td>
</tr>
</tbody>
</table>

Note: JavaScript also supports standard objects with the same type names as the primitive data types, for example, a Number object. These objects serve as wrappers for the corresponding primitive types and provide additional operations on these primitive types.
Table 7 shows how the AVM maps ABL data types to JavaScript data types. Note that an ABL data element of any primitive type set to the Unknown value (?) maps to the JavaScript `null` value. The ABL BLOB and CLOB are only allowed as fields of temp-table parameters, and their respective equivalents, MEMPTR and LONGCHAR, are only allowed as scalar parameters or as elements of Array parameters.

Table 7: ABL to JavaScript data type mappings  

<table>
<thead>
<tr>
<th>ABL data type</th>
<th>JavaScript data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLOB(^1)</td>
<td>String (Base64 encoded)</td>
</tr>
<tr>
<td>CHARACTER</td>
<td>String</td>
</tr>
<tr>
<td>CLOB(^1)</td>
<td>String</td>
</tr>
<tr>
<td>COM-HANDLE</td>
<td>Number</td>
</tr>
<tr>
<td>DATASET(^2)</td>
<td>An Object that maps to a ProDataSet and contains one or more Object instances, each of which maps to an ABL temp-table in the ProDataSet (see TEMP-TABLE)</td>
</tr>
<tr>
<td>DATE</td>
<td>String (ISO 8601 formatted string of the form “yyyy-mm-dd”)</td>
</tr>
<tr>
<td>DATETIME</td>
<td>String (ISO 8601 formatted string of the form “yyyy-mm-ddThh:mm:ss.sss”)</td>
</tr>
<tr>
<td>DATETIME-TZ</td>
<td>String (ISO 8601 formatted string of the form “yyyy-mm-ddThh:mm:ss.sss+hh:mm”)</td>
</tr>
<tr>
<td>DECIMAL</td>
<td>Number</td>
</tr>
<tr>
<td>primitive EXTENT(^2)</td>
<td>Where primitive is an ABL primitive data type (not a DATASET or TEMP-TABLE), maps to an Array of the JavaScript primitive data type that maps to the corresponding ABL primitive data type</td>
</tr>
<tr>
<td>HANDLE</td>
<td>Number</td>
</tr>
<tr>
<td>INT64</td>
<td>Number</td>
</tr>
<tr>
<td>INTEGER</td>
<td>Number</td>
</tr>
<tr>
<td>LOGICAL</td>
<td>Boolean (true or false)</td>
</tr>
<tr>
<td>LONGCHAR(^3)</td>
<td>String</td>
</tr>
<tr>
<td>MEMPTR(^3)</td>
<td>String (Base64 encoded)</td>
</tr>
<tr>
<td>RAW</td>
<td>Not supported</td>
</tr>
<tr>
<td>RECID</td>
<td>Not supported</td>
</tr>
<tr>
<td>ABL data type</td>
<td>JavaScript data type</td>
</tr>
<tr>
<td>--------------</td>
<td>----------------------</td>
</tr>
<tr>
<td>ROWID</td>
<td>String (Base64 encoded)</td>
</tr>
<tr>
<td>TEMP-TABLE²</td>
<td>An Object that contains a single Array of Object instances, where each Object in the Array maps to a record in the corresponding temp-table</td>
</tr>
</tbody>
</table>

1. In temp-tables only.
2. ABL ProDataSets, temp-tables, and arrays map to JavaScript objects and arrays using a structure that is identical to the OpenEdge-supported mapping to JSON objects and arrays. JSON (JavaScript Object Notation) is a character representation of JavaScript data types that is often used as a lightweight alternative to XML. For more information on ABL support for JSON, including the JSON representation of these ABL data types, see OpenEdge Development: Working with JSON.
3. As scalar parameters or elements of Array parameters only.
OpenEdge JavaScript Class and Object Reference

This appendix describes the JavaScript classes and objects installed with OpenEdge to support access to OpenEdge Mobile services from OpenEdge Mobile Apps. Each class or object description lists the documented class or object members by category: properties, methods, and events, with a short description of each member. For a detailed description of each member, see Appendix C, “OpenEdge JavaScript Properties, Methods, and Events Reference.”

Notes: JavaScript is a case-sensitive language. So, class type, property, method, and event names, as well as other defined language elements (such as data types) must have the specified letter case.

In addition to the documented members, these classes and objects might also contain undocumented members that are reserved for internal use by OpenEdge.

The OpenEdge JavaScript classes and objects for Mobile App development include:

- JSRecord object
- progress.data.JSDO class
- progress.data.Session class
- progress.ui.UIHelper class
- request object
JSRecord object

JSRecord is a JavaScript object that returns a record instance for any temp-table stored in the local storage of an associated progress.data.JSDO class instance (JSDO).

Properties

<table>
<thead>
<tr>
<th>Table 8: JSRecord object properties</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Member</strong></td>
</tr>
<tr>
<td>data property</td>
</tr>
</tbody>
</table>

Methods

<table>
<thead>
<tr>
<th>Table 9: JSRecord object methods</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Member</strong></td>
</tr>
<tr>
<td>assign( ) method (JSDO class)</td>
</tr>
<tr>
<td>getId( ) method</td>
</tr>
<tr>
<td>remove( ) method</td>
</tr>
</tbody>
</table>

Events

This object has no events.

Example

The following example assumes that a JSDO is referenced by the jsdo variable, and that a UITHelper instance associated with that JSDO is referenced by the uihelper variable. The example creates a new record object and displays it, along with a message with credit information using properties of the record object:

```javascript
function addRecord() {
    var jsrecord = jsdo.add({Balance: 10000, State: 'MA'});
    uihelper.display();
    alert('Record ID: ' + jsrecord.getId() + ' CreditLimit: ' + jsrecord.data.CreditLimit);
}
```

Note

Using the add( ), find( ), findById( ), or foreach( ) method, or the record property, on a given JSDO and table reference, a JSRecord instance returns a working record for the temp-table referenced in JSDO local storage. You can then use properties and methods of the JSRecord to update, delete, or display the specified record from the JSDO.

See also

add( ) method, find( ) method, findById( ) method, foreach( ) method, progress.data.JSDO class, record property, table reference property (JSDO)
progress.data.JSDO class

The progress.data.JSDO is a JavaScript class that provides access to ABL application services as OpenEdge Mobile resources. A single progress.data.JSDO object (JSDO) provides access to one Mobile resource in an OpenEdge Mobile service. A Mobile resource maps to an ABL singleton procedure or class running on an OpenEdge AppServer. The JSDO provides application-oriented, JavaScript methods to invoke the internal procedures, user-defined functions, or methods (ABL routines) of the corresponding singleton procedure or class. A Mobile resource maps specified ABL routines as one of several supported Mobile operation types, each of which corresponds to a particular method of the JSDO.

You identify how a JSDO maps JavaScript methods to operations of a given Mobile resource by adding annotations to the singleton procedure or class source code. These annotations define a Mobile interface for accessing the procedure or class as a Mobile resource. You can define a Mobile interface using features of Progress Developer Studio for OpenEdge (Developer Studio) in two ways: 1) when creating an ABL Business entity class, which supports common business operations on data models, and 2) for any existing ABL singleton class or procedure, whether or not the singleton accesses a separate Business entity object.

You can also use Developer Studio to define a Mobile resource as part of a Mobile service, and to generate the Mobile service together with the client-side artifacts required to create a corresponding JSDO for the resource. These client-side artifacts include a JSDO catalog file that identifies how a JSDO that you create can access the corresponding Mobile resource using methods of the JSDO.

At run time, the JSDO maintains local storage for managing temp-table data that is exchanged between the AppServer and JavaScript client, and it provides methods to read and write the data in JSDO local storage as individual JavaScript record objects. To support this data exchange, a Mobile resource can be organized into basic operation types that include built-in create, read, update, and delete (CRUD) operations, and non-built-in invoke operations. The built-in Mobile operations can operate on a single temp-table or on a single ProDataSet containing one or more temp-tables. Each built-in operation type maps to a corresponding built-in method of the JSDO. The records of each temp-table are presented as an array of record objects, which the built-in methods use to exchange the data with the AppServer. The built-in methods, through their corresponding operation types, support the common business operations that can be generated directly from an ABL Business entity. Other methods of the JSDO provide access to individual record objects of JSDO local storage. Based on the results of its methods, the JSDO also maintains a working record for each temp-table in its local storage that you can access directly using table and field references (see the notes). Thus, using the methods of a JSDO and its table references, you can interact with a corresponding Mobile resource in a consistent manner from one resource (and its corresponding JSDO) to another.

A JSDO also supports non-built-in invoke operations that allow specific ABL routines to be exposed in a Mobile resource and executed as corresponding JavaScript methods. You can do this in Developer Studio by annotating ABL routines specifically as invoke operations. You can then call each ABL routine annotated as an invoke operation using a unique invocation method on the JSDO. Note that data exchanged between the AppServer and client using invoke operations is not automatically stored in JSDO local storage. It is initially accessible only through parameters and return values of the invocation methods provided by the JSDO. You can subsequently use JSDO methods for accessing JSDO local storage to exchange data between the invocation methods and local storage, which is maintained and synchronized with the AppServer using the JSDO built-in methods.
When you instantiate a JSDO, it relies on a prior login session that you can establish using an instance of the progress.data.Session class. This login session enables optionally secure communications between the client JSDO and the Web server, specified Mobile services, and ultimately the AppServer that implements the Mobile resource accessed by the JSDO.

### Constructors

Two constructors are available for the JSDO. The first constructor takes the name of the corresponding Mobile resource as a parameter; the second constructor takes an initialization object as a parameter:

#### Syntax

```plaintext
progress.data.JSDO ( resource-name )
progress.data.JSDO ( init-object )
```

**resource-name**

A string expression set to the name of a resource provided by a Mobile service for which a login session has been started.

**init-object**

An object that can contain any writable JSDO properties. It must contain the required JSDO `name` property, which specifies the Mobile resource for the JSDO. It can also contain either or both of the following initialization properties:

- **autoFill** — A Boolean that specifies whether the the JSDO invokes its `fill( )` method upon instantiation to initialize its local storage with data from the Mobile resource. The default value is `false`.
- **events** — An object that specifies one or more JSDO event subscriptions, each with its properties represented as an array, with the following syntax:

#### Syntax

```plaintext
events : {
  'event' : [ {  
    [ scope : object-ref , ]
    fn : function-ref
  } ] ,
  'event' : [ {  
    [ scope : object-ref , ]
    fn : function-ref
  } ] ...  
}
```

**event**

The name of an event to which the JSDO instance subscribes. See Table 13 for a list of available JSDO events.
object-ref

An optional object reference that defines the execution scope of the function called when the event fires. If the `scope` property is omitted, the execution scope is the global object (usually the browser or device window).

function-ref

A reference to an event handler function that is called when the event fires.

Each event passes a fixed set of parameters to its event handler, as described in the reference entry for the event in Appendix C, “OpenEdge JavaScript Properties, Methods, and Events Reference.”

The resource name specified for the constructor must match the name of a Mobile resource provided by a Mobile service for which a login session has already been started. After the JSDO is created, it uses the information stored in the JSDO catalog that is loaded for the Mobile service to communicate with the specified resource.

Example

The following example illustrates the use of an initialization object to instantiate a JSDO:

```javascript
dsItems = new progress.data.JSDO({
  name : 'Item',
  autoFill : false,
  events : {
    'afterFill' : [ {
      scope : this,
      fn : function (jsdo, success, request) {
        // afterFill event handler statements ...
      }
    }
  }
});
```

Properties

Table 10: progress.data.JSDO properties

<table>
<thead>
<tr>
<th>Member</th>
<th>Brief description (See also the reference entry)</th>
</tr>
</thead>
<tbody>
<tr>
<td>autoSort property</td>
<td>A Boolean on a JSDO and its table references that indicates if record objects are sorted automatically on the affected table references in JSDO local storage at the completion of a supported JSDO operation</td>
</tr>
<tr>
<td>caseSensitive property</td>
<td>A Boolean on a JSDO and its table references that indicates if String field comparisons performed by supported JSDO operations are case sensitive or case-insensitive for the affected table references in JSDO local storage</td>
</tr>
<tr>
<td>name property</td>
<td>The name of the Mobile resource for which the current JSDO is created</td>
</tr>
</tbody>
</table>
Methods

Certain methods of the progress.data.JSDO class are called on the JSDO object itself, without regard to a table reference, whether that reference is explicit (specified in the method signature) or implicit (in the case of a JSDO containing only one temp-table that is not explicitly specified). Other methods can be called on a reference to a temp-table mapped by the Mobile resource for which the current JSDO is created.

Table 10: progress.data.JSDO properties

<table>
<thead>
<tr>
<th>Member</th>
<th>Brief description (See also the reference entry)</th>
</tr>
</thead>
<tbody>
<tr>
<td>record property</td>
<td>A property on a JSDO table reference that references a JSRecord object with the data for the working record of a temp-table referenced in JSDO local storage</td>
</tr>
<tr>
<td>table reference property (JSDO)</td>
<td>An object reference property on the JSDO that has the name of a temp-table mapped by the Mobile resource for which the current JSDO is created</td>
</tr>
<tr>
<td>useRelationships property</td>
<td>A Boolean on the JSDO that specifies whether JSDO methods that operate on table references in JSDO local storage work with the table relationships defined in the schema (that is, work only on the records of a child table that are related to the parent)</td>
</tr>
</tbody>
</table>

Table 11: progress.data.JSDO class-instance methods

<table>
<thead>
<tr>
<th>Member</th>
<th>Brief description (See also the reference entry)</th>
</tr>
</thead>
<tbody>
<tr>
<td>addRecords( ) method</td>
<td>Reads a specified object and updates the local storage of the JSDO</td>
</tr>
<tr>
<td>fill( ) method</td>
<td>Initializes the JSDO local storage from the data records in a single temp-table, or in one or more temp-tables of a ProDataSet, as returned by the built-in read operation of the Mobile resource for which the JSDO is created</td>
</tr>
<tr>
<td>invocation method</td>
<td>Any method on the JSDO that is defined by the Mobile resource to execute a corresponding ABL routine on the AppServer as an invoke operation</td>
</tr>
<tr>
<td>saveChanges( ) method</td>
<td>Synchronizes to the AppServer all changes made to JSDO local storage since the last call to the fill( ) or saveChanges( ) method</td>
</tr>
<tr>
<td>subscribe( ) method</td>
<td>Subscribes a given event handler function to a named event for a JSDO or table reference</td>
</tr>
<tr>
<td>unsubscribe( ) method</td>
<td>Unsubscribes a given event handler function from a named event of a JSDO or table reference</td>
</tr>
</tbody>
</table>
Table 12:  progress.data.JSDO table-reference methods

<table>
<thead>
<tr>
<th>Member</th>
<th>Brief description (See also the reference entry)</th>
</tr>
</thead>
<tbody>
<tr>
<td>add( ) method</td>
<td>Creates a new record object for a temp-table referenced in JSDO local storage and returns a reference to the new record</td>
</tr>
<tr>
<td>addRecords( ) method</td>
<td>Reads a specified object and updates the local storage of the JSDO</td>
</tr>
<tr>
<td>assign( ) method (JSDO class)</td>
<td>Updates field values for the specified temp-table record referenced in JSDO local storage</td>
</tr>
<tr>
<td>find( ) method</td>
<td>Searches for a record in a temp-table referenced in JSDO local storage and returns a reference to that record if found</td>
</tr>
<tr>
<td>findByld( ) method</td>
<td>Locates and returns the record in JSDO local storage with the internal ID you specify</td>
</tr>
<tr>
<td>foreach( ) method</td>
<td>Loops through the records of a temp-table referenced in JSDO local storage and invokes a function as a parameter on each iteration</td>
</tr>
<tr>
<td>getData( ) method</td>
<td>Returns an array of record objects for a temp-table referenced in JSDO local storage</td>
</tr>
<tr>
<td>getId( ) method</td>
<td>Returns the unique internal ID for the specified temp-table record referenced in JSDO local storage</td>
</tr>
<tr>
<td>getSchema( ) method</td>
<td>Returns an array of objects, one for each field that defines the schema of a temp-table referenced in JSDO local storage</td>
</tr>
<tr>
<td>remove( ) method</td>
<td>Deletes the specified temp-table record referenced in JSDO local storage</td>
</tr>
<tr>
<td>setSortFields( ) method</td>
<td>Specifies or clears the record fields on which to automatically sort the record objects for a table reference after you have set its autoSort property to true (the default)</td>
</tr>
<tr>
<td>setSortFn( ) method</td>
<td>Specifies or clears a user-defined sort function with which to automatically sort the record objects for a table reference after you have set its autoSort property to true (the default)</td>
</tr>
<tr>
<td>sort( ) method</td>
<td>Sorts the existing record objects for a table reference in JSDO local storage using either specified sort fields or a specified user-defined sort function</td>
</tr>
<tr>
<td>subscribe( ) method</td>
<td>Subscribes a given event handler function to a named event for a JSDO or table reference</td>
</tr>
<tr>
<td>unsubscribe( ) method</td>
<td>Unsubscribes a given event handler function from a named event of a JSDO or table reference</td>
</tr>
</tbody>
</table>
### Table 13: progress.data.JSDO events

<table>
<thead>
<tr>
<th>Member</th>
<th>Brief description (See also the reference entry)</th>
</tr>
</thead>
<tbody>
<tr>
<td>afterCreate event</td>
<td>Fires after the JSDO, by means of a <code>saveChanges()</code> call following an <code>add()</code> call, sends a request to create a record and receives a response to this request from the AppServer</td>
</tr>
<tr>
<td>afterDelete event</td>
<td>Fires after the JSDO, by means of a <code>saveChanges()</code> call following a <code>remove()</code> call, sends a request to delete a record and receives a response to this request from the AppServer</td>
</tr>
<tr>
<td>afterFill event</td>
<td>Fires after the JSDO, by means of a <code>fill()</code> call, sends a request to read a temp-table or ProDataSet into JSDO local storage and receives a response to this request from the AppServer</td>
</tr>
<tr>
<td>afterInvoke event</td>
<td>Fires after a non–built-in method is called asynchronously on a JSDO and a response to the request is received from the AppServer</td>
</tr>
<tr>
<td>afterSaveChanges event</td>
<td>Fires once for each call to the <code>saveChanges()</code> method on a JSDO, after responses to all create, update, and delete requests have been received from the AppServer</td>
</tr>
<tr>
<td>afterUpdate event</td>
<td>Fires after the JSDO, by means of a <code>saveChanges()</code> call following an <code>assign()</code> call, sends a request to update a record and receives a response to this request from the AppServer</td>
</tr>
<tr>
<td>beforeCreate event</td>
<td>Fires before the JSDO, by means of a <code>saveChanges()</code> call following an <code>add()</code> call, sends a request to create a record</td>
</tr>
<tr>
<td>beforeDelete event</td>
<td>Fires before the JSDO, by means of a <code>saveChanges()</code> call following a <code>remove()</code> call, sends a request to delete a record</td>
</tr>
<tr>
<td>beforeFill event</td>
<td>Fires before the JSDO, by means of a <code>fill()</code> call, sends a request to the AppServer to read a temp-table or ProDataSet into JSDO local storage</td>
</tr>
<tr>
<td>beforeInvoke event</td>
<td>Fires when a non–built-in method is called asynchronously on a JSDO, before the request for the operation is sent to the AppServer</td>
</tr>
<tr>
<td>beforeSaveChanges event</td>
<td>Fires once for each call to the <code>saveChanges()</code> method on a JSDO, before any create, update, or delete requests are sent to the AppServer</td>
</tr>
<tr>
<td>beforeUpdate event</td>
<td>Fires before the JSDO, by means of a <code>saveChanges()</code> call following an <code>assign()</code> call, sends a request to the AppServer to update a record</td>
</tr>
</tbody>
</table>
The JSDO can subscribe to the events listed in Table 13 in either of two ways:

- **Subscription via JSDO constructor** — In the `init-object` parameter of the constructor, list each subscribed event with an optional scope object and an event handler method to be executed when the event fires. See the “Constructors” section on page 192.

- **Subscription via `subscribe()` method** — See the “`subscribe()` method” section on page 305.

**Note:** JSDO events do not fire if the method call is synchronous.

**Example**

The following example reads customer records from a server-side temp-table or ProDataSet and displays fields from the records in a list on the current Web page:

### Table 14: Example — Using an OpenEdge JSDO

```javascript
var session = new progress.data.Session();
  // assuming userName and password came from the UI
  var loginResult = session.login('/MobileApp', userName, password);
  if (loginResult !== progress.data.Session.LOGIN_SUCCESS) {
    // process login failure here and return/throw error, etc
    throw new Error("Login failed");
  }
  session.addCatalog('/MobileApp/static/mobile/MobileSvc.json');

var jsdoOrderEntry = new progress.data.JSDO('OrderEntry');
jsdoOrderEntry.subscribe('AfterFill', onAfterFill);
var uihelper = new progress.ui.UIHelper({ jsdo: dataSet });

// set list view and detail page for displaying data
uihelper.eCustomer.setDetailPage({ name: 'custdetail' });
uihelper.eCustomer.setListView({
  name: 'listview',
  format: '{CustNum}<br>{Name}<br>{State}',
  autoLink: true
});

function onAfterFill() {
  uihelper.eCustomer.clearItems();
  jsdoOrderEntry.eCustomer.foreach(function(jsrecord) {
    uihelper.eCustomer.addItem();
  });
  uihelper.eCustomer.showListView();
}
```
progress.data.JSDO class

Notes

- For more information on defining a Mobile interface, including the built-in CRUD and non-built-in invoke operations of a Mobile resource, see Chapter 3, "Creating Mobile Services."

- The JSDO supports a working record for each temp-table referenced in its local storage. Certain methods set a specific record as the working record. After other methods execute, there is no working record or existing working records remain unchanged. When there is a working record, you can access the fields of the record using one of the following mechanisms:

Syntax

```javascript
jsdo.table-ref.field-ref
jsdo.record.data.field-ref // Read only when a single table-ref is defined
jsdo.table-ref.record.data.field-ref // Read only
jsrecord-ref.data.field-ref // Read only
```

`jsdo`

The reference to a JSDO, and if the JSDO has only one temp-table, an implied reference to the working record defined for that temp-table.

`table-ref`

A table reference with the name of a temp-table in jsdo local storage and a reference to the temp-table working record. There is one table reference in a JSDO for each temp-table referenced by the JSDO.

`field-ref`

A field reference on a table-ref, or a property on the data property object, with the name and value of a field in the working record of the referenced temp-table. There is one such field reference and data object property for each field defined in the temp-table schema.

`record`

A property of type JSRecord on a table reference, which references the working record of a temp-table specified by:

- jsdo.table-ref

  - jsdo if the JSDO has only one temp-table

If the JSDO has more than one temp-table, the record property is null at the JSDO level and is available only on a table-ref.

`data`

A property on a JSRecord object with the field values for the working record specified by:

- jsdo.table-ref

  - jsdo if the JSDO has only one temp-table

  - js-record of an associated JSDO temp-table
Note: If a field-ref has the same name as a built-in property or method of the JSDO, you must use the data property to reference its value in the working record.

Caution: Never write directly to a field-ref using this data property; in this case, use field-ref only to read the data. Writing to data using such a reference does not mark the record for update when calling the saveChanges() method, nor does it re-sort the record in local storage according to any order you have established using the autoSort property. To mark a record for update and automatically re-sort the record according to the autoSort property, you must assign a field value either by setting a jsdo.table-ref.field-ref for a working record or by calling the assign() method on a valid table-ref or JSRecord object reference.

jsrecord-ref

A reference to a JSRecord object of a temp-table in JSDO local storage. You can return a jsrecord-ref for a working record as the value of the record property or as a value returned by supported JSDO built-in methods that return a working record, such as add() and find().

For more information on the properties available to reference working record fields using this syntax, see Table 10 and Table 8. For more information on the methods for setting the working record for referenced temp-tables, see Table 11 and Table 9.

- Many JSDO built-in methods are actually invoked on a JSDO table reference, and can only be invoked on the JSDO itself when its local storage is initialized with a single temp-table.

- For a multi-table ProDataSet, the JSDO accesses the data for all unrelated temp-tables in local storage as top-level tables of the JSDO. Access to data for all related child temp-tables depends on the working record of the parent temp-table in the JSDO and the setting of the useRelationships property.

See also JSRecord object, progress.data.Session class, record property, table reference property (JSDO), table reference property (UIHelper)
The `progress.data.Session` is a JavaScript class that can manage user authentication and session identification information in HTTP/S messages sent between `progress.data.JSDO` objects (JSDOs) running in an OpenEdge Mobile App and Mobile services running on a Web server. The authentication information includes a user ID and password (`user credentials`). The session identification information includes a URI, which identifies the Mobile Web application that provides the REST transport between its defined set of Mobile services and the client that accesses them, and a session ID, which identifies the user login session for the entire set of Mobile services supported by the Mobile Web application.

To start a user login session, you can either allow the Web browser or hybrid application wrapper to complete the user login process or invoke the `login()` method on a `Session` object that you have instantiated, passing as parameters the Mobile Web application URI, optional user credentials, and an optional specified Mobile Web application resource (such as a static HTML page) to authenticate access. Once started, a login session for a Mobile Web application supports all Mobile services that the application provides, each of which can provide one or more Mobile resources.

Each Mobile service provided by a Mobile Web application relies on a separate JSDO catalog file to define the communications between its Mobile resources and the JSDOs that access them from the client. Once a user login session is established for a Mobile Web application, you can use its `Session` object to load the catalog for each Mobile service provided by the Web application. Once the JSDO catalog is loaded for the Mobile service, you can instantiate a JSDO to access any Mobile resource provided by the service in the catalog. If required, the authentication information for the session is also used to authorize access to the Mobile resource by its JSDO.

All JSDOs can thus rely on a single `Session` object to manage the user login session for all Mobile services and their resources provided by a single Mobile Web application. This single `Session` object then manages the session life cycle from startup (login) to shutdown (logout) for all JSDOs of a Mobile App and the Mobile services they access from the same Mobile Web application.

**Constructor**

```javascript
progress.data.Session ( )
```

Instantiates a `Session` object that you can use to start a user login session for a Mobile Web application and load the JSDO catalog for each supported Mobile service whose resources are accessed using JSDOs.
<table>
<thead>
<tr>
<th>Member</th>
<th>Brief description (See also the reference entry)</th>
</tr>
</thead>
<tbody>
<tr>
<td>authenticationModel property</td>
<td>A string constant that specifies the type of authentication that the server requires from the client.</td>
</tr>
<tr>
<td>catalogURIs property</td>
<td>Returns the list of URIs used to load the JSDO catalogs to access the Mobile services provided by the Mobile Web application for which the current Session object manages a user login session.</td>
</tr>
<tr>
<td>clientContextId property</td>
<td>The value of the most recent client context identifier (CCID) that the Session object has found in the X-CLIENT-CONTEXT-ID HTTP header of a server response message.</td>
</tr>
<tr>
<td>lastSessionXHR property</td>
<td>Returns an object reference to the XMLHttpRequest object (XHR) that was most recently used by the progress.data.Session object to execute a Session object method.</td>
</tr>
<tr>
<td>loginHttpStatus property</td>
<td>Returns the specific HTTP status code returned in the response from the most recent login attempt on the current Session object.</td>
</tr>
<tr>
<td>loginResult property</td>
<td>Returns the return value of the login() method, which is the basic result code for the most recent login attempt on the current Session object.</td>
</tr>
<tr>
<td>loginTarget property</td>
<td>Returns the string appended to the Mobile Web application URI passed to the login() method to form the URI of an application resource against which the user has been authenticated for the current login session.</td>
</tr>
<tr>
<td>onOpenRequest property</td>
<td>Returns a developer-specified callback function that the Session object executes to modify a request object before sending the request object to the server.</td>
</tr>
<tr>
<td>services property</td>
<td>Returns an array of objects that identifies the Mobile services that have been loaded for the current Session object and its Mobile Web application.</td>
</tr>
<tr>
<td>serviceURI property</td>
<td>Returns the URI to the Mobile Web application passed as a parameter to the most recent call to the login() method on the current Session object, whether or not the most recent call to login() succeeded.</td>
</tr>
<tr>
<td>userName property</td>
<td>Returns the user ID passed as a parameter to the most recent call to the login() method on the current Session object.</td>
</tr>
</tbody>
</table>
progress.data.Session class

Methods

<table>
<thead>
<tr>
<th>Member</th>
<th>Brief description (See also the reference entry)</th>
</tr>
</thead>
<tbody>
<tr>
<td>addCatalog( ) method</td>
<td>Loads the JSDO catalog for a login session established using the login( ) method</td>
</tr>
<tr>
<td>login( ) method</td>
<td>Starts a user login session on the current Session object by sending an HTTP request with user credentials to a URI for a specified Mobile Web application</td>
</tr>
<tr>
<td>logout( ) method</td>
<td>Terminates the login session on the Web server associated with the Session object</td>
</tr>
</tbody>
</table>

Events
This class has no events.

Example
This is an example of how you might create a Session object and use the URI to a Mobile Web application to log into the application, load the JSDO catalog for a Mobile service provided by that application, and create a JSDO for a Customer Mobile resource defined by that service in the catalog:

```javascript
// create Session
pdsession = new progress.data.Session();

// log in, i.e., authenticate to the Mobile Web application
pdsession.login('/SportsApp', username, password);

// load catalog for a service that's part of the Mobile Web application
pdsession.addCatalog('/SportsApp/static/mobile/OrderEntrySvc.json');

// create JSDO
customers = new progress.data.JSDO({ name: 'Customer' });

/* etc. - additional code to fill and use the JSDO */
```

The JSDO automatically finds and uses the Session object on which a catalog that defines the Customer resource is loaded.

Notes

- Use an instance of this class to call the login( ) method to start a user login session, call the addCatalog( ) method to load one or more JSDO catalogs for the session, and possibly call the logout( ) method to terminate the session. To use the same Session instance to start a new login session, you must call the logout( ) method first.

- The behavior of a login session using this class depends on the authentication model of the Web server and how its resources are protected and accessed. For more information, see the description of the login( ) method.

- If you have special requirements for sending Mobile requests to the Web server, you can modify the XMLHttpRequest object that is sent by the Session object. To do so, assign a callback function as the value of Session.onOpenRequest.

See also
addCatalog( ) method, login( ) method, logout( ) method, progress.data.JSDO class
progress.ui.UIHelper class

The progress.ui.UIHelper class is an OpenEdge class that provides methods for managing the user interface of a OpenEdge Mobile App. This class is intended for use by:

- Developers using JavaScript libraries, such as jQuery Mobile, to build user interfaces by creating HTML-based lists and fields.

- Developers using OpenEdge Mobile App Builder without JSDO Services. JSDO Services eliminate the need for UIHelper functionality.

Each instance of UIHelper supports the display of data for a specific JSDO, and typically controls the format and content of a list view (showing items representing table records) and a detail page (showing a form with input fields for the list item clicked by the user).

Constructor

Syntax

progress.ui.UIHelper( JSDO-object )

Instantiates a UIHelper object for use in managing the UI for a specified JSDO.

JSDO-object

An object reference to the JSDO associated with the UIHelper instance.

Properties

Table 18: progress..ui.UIHelper properties

<table>
<thead>
<tr>
<th>Member</th>
<th>Brief description (See also the reference entry)</th>
</tr>
</thead>
<tbody>
<tr>
<td>table reference property (UIHelper)</td>
<td>An object reference property on a UIHelper instance that corresponds to a temp-table of the JSDO with which the instance is associated</td>
</tr>
</tbody>
</table>

Methods

Table 19: progress.ui.UIHelper class-level methods

<table>
<thead>
<tr>
<th>Member</th>
<th>Brief description (See also the reference entry)</th>
</tr>
</thead>
<tbody>
<tr>
<td>setFieldTemplate( ) method</td>
<td>Specifies the format or all detail forms created during the JavaScript session</td>
</tr>
<tr>
<td>setItemTemplate( ) method</td>
<td>Specifies the format for items in all list views created during the JavaScript session</td>
</tr>
</tbody>
</table>
### Table 20: progress.ui.UIHelper table-reference methods

<table>
<thead>
<tr>
<th>Member</th>
<th>Brief description (See also the reference entry)</th>
</tr>
</thead>
<tbody>
<tr>
<td>addItem( ) method</td>
<td>Adds an item based on the working record to a list view</td>
</tr>
<tr>
<td>assign( ) method (UIHelper class)</td>
<td>Updates the working record in JSDO local storage with the values currently displayed on the detail page form</td>
</tr>
<tr>
<td>clearItems( ) method</td>
<td>Clears the items from a list view</td>
</tr>
<tr>
<td>display( ) method</td>
<td>Copies the field values of a given record to corresponding fields in the current HTML document for display in the form on the detail page</td>
</tr>
<tr>
<td>getFormFields( ) method</td>
<td>Reads the schema of the specified table and returns HTML text that defines corresponding input fields to appear on the form on the detail page</td>
</tr>
<tr>
<td>getFormRecord( ) method</td>
<td>Returns a JSRecord object for the record shown on the form on the detail page</td>
</tr>
<tr>
<td>getListViewRecord( ) method</td>
<td>Returns a JSRecord object for a specified item in a list view</td>
</tr>
<tr>
<td>setDetailPage( ) method</td>
<td>Specifies the HTML page that contains the form in which item details are displayed</td>
</tr>
<tr>
<td>setListView( ) method</td>
<td>Defines a list view for a given table</td>
</tr>
</tbody>
</table>

### Events

This class has no events.

### Example

The sample Mobile app shown below is followed by the code for its index page (index.html) and client logic (customers.js):

![Sample Mobile App](image)

```html
<!-- index.html -->
```

```javascript
// customers.js
```
Table 21: Example — index.html

```html
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <title>Customers</title>
    <link rel="stylesheet" href="http://code.jquery.com/mobile/1.1.0/jquery.mobile-1.1.0.min.css" />
    <style>
      /* App custom styles */
    </style>
    <script src="http://ajax.googleapis.com/ajax/libs/jquery/1.7.1/jquery.min.js"></script>
    <script src="http://code.jquery.com/mobile/1.1.0/jquery.mobile-1.1.0.min.js"></script>
    <script src="../progress.session.js"></script>
    <script src="../progress.js"></script>
    <script src="customers.js"></script>
  </head>
  <body>
    <div data-role="page" id="custlist">
      <div data-theme="b" data-role="header">
        <h3>Customers</h3>
      </div>
      <div data-role="content">
        <ul id="listview" data-role="listview" data-divider-theme="b" data-inset="true" data-filter="true">
        </ul>
      </div>
    </div>
    <div data-role="page" id="custdetail" data-add-back-btn="true">
      <div data-role="header"><h1>Customer</h1></div>
      <div data-role="content">
        <form action="" id="customerform">
        </form>
      </div>
    </div>
  </body>
</html>
```
Table 22: Example — customers.js

```javascript
var customers;
var uihelper;
var forminitialized = false;

$(document).ready(function() {
    var session = new progress.data.Session();
    customers = new progress.data.JSDO({ name: 'CustomerOrder' });
    customers.subscribe('AfterFill', onAfterFillCustomers,
    this);
    uihelper = new progress.ui.UIHelper({ jsdo: customers });
    uihelper.eCustomer.setDetailPage({
        name: 'custdetail'
    });
    uihelper.eCustomer.setListView({
        name: 'listview',
        format: '{CustNum}<br>{Name}<br>{State}',
        autoLink: true
    });
    customers.fill();
});

function onAfterFillCustomers() {
    uihelper.eCustomer.clearItems();
    customers.eCustomer.foreach(function(customer) {
        uihelper.eCustomer.addItem();
    });
    uihelper.eCustomer.showListView();
}
```

Notes

- See also progress.data.JSDO class
request object

An object containing data and status information pertaining to a call to one of the methods of an associated progress.data.JSDO class instance (JSDO) that executes a Mobile built-in or invoke operation on the AppServer. The request object is returned by the method call. In the case of an asynchronous call, the request object is passed as a parameter to the event handler function defined on the JSDO.

Properties

Table 23: Request object properties

<table>
<thead>
<tr>
<th>Member</th>
<th>Brief description (See also the reference entry)</th>
</tr>
</thead>
<tbody>
<tr>
<td>async property</td>
<td>A Boolean that indicates, if set to true, that the Mobile operation was executed asynchronously on the client</td>
</tr>
<tr>
<td>batch property</td>
<td>A reference to an object with a property named operations, which is an array containing the request objects for each of the several operations performed in response to a call to the JSDO saveChanges( ) method</td>
</tr>
<tr>
<td>fnName property</td>
<td>For an invoke operation, the name of the JSDO invocation method that called the operation.</td>
</tr>
<tr>
<td>jsdo property</td>
<td>An object reference to the JSDO that performed the operation returning the request object</td>
</tr>
<tr>
<td>jsrecord property</td>
<td>An object reference to the record created, updated, or deleted by the operation</td>
</tr>
<tr>
<td>paramObj property</td>
<td>A reference to the object, if any, that was passed as a parameter to the method that returned the request object</td>
</tr>
<tr>
<td>response property</td>
<td>Returns an object whose properties contain data from a Mobile built-in or invoke operation executed on the AppServer</td>
</tr>
<tr>
<td>success property</td>
<td>A Boolean that indicates, if set to true, that the Mobile operation was successfully executed</td>
</tr>
<tr>
<td>xhr property</td>
<td>A reference to the XMLHttpRequest object used to perform the request</td>
</tr>
</tbody>
</table>

Methods

This object has no methods.

Events

This object has no events.

See also fill( ) method, invocation method, progress.data.JSDO class, saveChanges( ) method
request object
OpenEdge JavaScript Properties, Methods, and Events Reference

This appendix describes the following properties, methods, and events of OpenEdge JavaScript classes and objects:

- add() method
- addCatalog() method
- addItem() method
- addRecords() method
- afterCreate event
- afterDelete event
- afterFill event
- afterInvoke event
- afterSaveChanges event
- afterUpdate event
- assign() method (JSDO class)
- assign() method (UIHelper class)
- async property
- authenticationModel property
- autoSort property
• batch property
• beforeCreate event
• beforeDelete event
• beforeFill event
• beforeInvoke event
• beforeSaveChanges event
• beforeUpdate event
• caseSensitive property
• catalogURIs property
• clearItems() method
• clientContextId property
• data property
• display() method
• fill() method
• find() method
• findById() method
• fnName property
• foreach() method
• getData() method
• getFormFields() method
• getFormRecord() method
• getId() method
• getSchema() method
• getListViewRecord() method
• invocation method
• jsdo property
• jsrecord property
• lastSessionXHR property
• login() method
• loginHttpStatus property
• loginResult property
• loginTarget property
• logout( ) method
• name property
• onOpenRequest property
• paramObj property
• record property
• remove( ) method
• response property
• saveChanges( ) method
• services property
• serviceURI property
• setDetailPage( ) method
• setFieldTemplate( ) method
• setItemTemplate( ) method
• setListView( ) method
• setSortFields( ) method
• setSortFn( ) method
• showListView( ) method
• sort( ) method
• subscribe( ) method
• success property
• table reference property (JSDO)
• table reference property (UIHelper)
• unsubscribe( ) method
• useRelationships property
• userName property
• xhr property
add( ) method

Creates a new record object for a temp-table referenced in JSDO local storage and returns a reference to the new record.

After completing execution, the new record becomes the working record for the associated temp-table. If the temp-table has child temp-tables, the working record for these child tables is not set. To synchronize the change on the AppServer, call the saveChanges( ) method.

**Return type:** JSRecord object  
**Applies to:** progress.data.JSDO class, table reference property (JSDO)

**Syntax**

```
[table-ref.]add ( [ new-record-object ] )
```

**table-ref**

A table reference on the JSDO. If the JSDO references only a single temp-table, the method can be called on the JSDO itself.

**new-record-object**

If specified as a non-null object, passes in the data to create the record for the JSRecord instance. The data to create the record is identified by one or more properties, each of which has the name of a corresponding field in the temp-table schema and has the value to set that field in the new table record.

If you omit or set the parameter to `null`, or you do not include properties of `new-record-object` for all fields in the new record, the method uses the default values from the temp-table schema stored in the catalog to set the unspecified record fields.

**Note:** After this method adds the new record object, and if you have set up automatic sorting using the `autoSort` property, all the record objects for the affected table reference are sorted accordingly. If the sorting is done using sort fields, any `String` fields are compared according to the value of the `caseSensitive` property.

If `table-ref` references a child temp-table in a ProDataSet, when the `useRelationships` property is true, `add( )` uses the relationship to set related field values of the new child record from the working record of the parent temp-table. However, if the working record of the parent is not set, `add( )` throws an error. If `useRelationships` is false, the fields for the new record are set as specified by `new-record-object` and no error is thrown.

For example, given a JSDO created for a ProDataSet resource with a `customer` and related child order temp-table, the `add( )` method in the following code fragment uses this relationship to automatically set the `CustNum` field in a new record added to the order table:
```javascript
var dataSet = new Progress.data.JSDO( 'CustomerOrderDS' );
dataSet.customer.add( { CustNum: 1000, Balance: 10000, State: 'MA' } );

// CustNum is set automatically by using the relationship
dataSet.order.add( { OrderNum: 1000 } );
```

**Note:** OpenEdge adds the new record object with an OpenEdge-reserved `Number` field named `_id`, which uniquely identifies the record in JSDO local storage. Note that once you have saved the new record object to the AppServer using `saveChanges()`, its `_id` value can change with each invocation of the `fill()` method. For more information on this value, see the description of the `getId()` method.

**See also:** `autoSort` property, `caseSensitive` property, `fill()` method, `getId()` method, `getSchema()` method, `data` property, `saveChanges()` method
addCatalog( ) method

Loads the JSDO catalog for a login session established using the login( ) method. If the login( ) method has not been called on the current Session object, or has been called but failed, the method throws an Error object.

Return type: Number
Applies to: progress.data.Session class

Syntax

```
addCatalog( catalog-uri [, cat-user-name [, cat-password ]] )
```

catalog-uri

The URI of a JSDO catalog file. The file is typically in a location relative to the Mobile Web application where the Session object has a login session.

If the Mobile App from which you are logging in is a Mobile Web App deployed to the same Apache Tomcat server as the Mobile Web application, you can specify catalog-uri as a relative URI, for example:

/SportsMobileApp/static/mobile/OrderEntrySvc.json, which is relative to the deployment end point (Tomcat server domain or host and port).

If the Mobile App from which you are logging in is a Mobile Native App that will be installed to run directly in a native device container, or if it is a Mobile Web App deployed to a different Web server from the Mobile Web application, you must specify catalog-uri as an absolute URI to the Tomcat server domain or host and port, for example,

http://www.progress.com/SportsMobileApp/static/mobile/OrderEntrySvc.json, or perhaps for testing,


Note: For any Mobile App that must specify catalog-uri as an absolute URI, you must maintain separate JavaScript sources to deploy the Mobile App for different Mobile Web application environments, such as one for testing and one for production.

Note: The default catalog URI for a catalog created for a Mobile service, relative to the Apache Tomcat server domain or host and port where the session is logged in, is the following:

/MobileWebApplicationName/static/mobile/ServiceName.json

Where MobileWebApplicationName is the name of the Mobile Web application and ServiceName is the name of the Mobile service for which the JSDO catalog is created.

cat-user-name

A string expression containing a user ID to authenticate access to a protected catalog. If you do not specify cat-user-name, catalog access is authorized using existing user credentials (if any).
**addCatalog( ) method**

**cat-password**

A string expression containing a password (if required) to authenticate the user specified by `cat-user-name`.

**Note:** Typically, you do not need to specify `cat-user-name` or `cat-password`. These optional parameters are available primarily if you store the catalog somewhere other than as part of the Mobile Web application where the user session is logged in.

You can read the `catalogURIs` property to return the URIs for all catalogs previously loaded for the login session.

When the method completes, it returns one of the following numeric constants to indicate the result:

- `progress.data.Session.SUCCESS` — The specified JSDO catalog loaded successfully.
- `progress.data.Session.AUTHENTICATION_FAILURE` — The catalog failed to load because of a user authentication error.
- `progress.data.Session.CATALOG_ALREADY_LOADED` — The specified JSDO catalog did not load because it is already loaded.

For all other errors, this method throws an `Error` object. For more detailed information about any response (successful or unsuccessful) returned from the Web server, you can also check the XMLHttpRequest object (XHR) returned by the `lastSessionXHR` property.

**See also:** `catalogURIs` property, `lastSessionXHR` property, `login( ) method`
addItem( ) method

Adds an item based on the working record to a list view. The appearance and content of the list view item are controlled by a specified template and format, respectively:

- The template is defined by `setItemTemplate( )`, optionally overridden for this list view by `setListView( )`. OpenEdge Mobile provides a default template suitable for use in a jQuery Mobile environment.
- The format is defined by `setListView( )`, optionally overridden for this item by the `format` parameter.

Return type: null
Applies to: progress.ui.UIHelper class, table reference property (UIHelper)

Syntax

```
[table-ref.]addItem ([ format ])
```

**table-ref**

A table reference on the UIHelper instance. If the JSDO associated with the UIHelper instance references only a single temp-table, the method can be called on the UIHelper instance itself.

**format**

A String value specifying the fields displayed for the list item, overriding the format defined by `setListView( )`. For example:

```
addItem ('{CustNum}<br>{Name}<br>{State}<br>{Country}')
```

See also: `setListView( ) method`, `setItemTemplate( ) method`
addRecords( ) method

Reads a specified object and updates the local storage of the JSDO. This updates all temp-tables read in for a ProDataSet or a specified temp-table, depending on how the method is called. The data is merged into JSDO local storage and affects existing data according to a specified merge mode and optional key fields.

After execution, the working record set for each JSDO table reference remains unchanged.

Return type: null

Applies to: progress.data.JSDO class, table reference property (JSDO)

Syntax

```
[table-ref.] addRecords ( merge-object, add-mode [ , key-fields ] )
```

**table-ref**

A table reference on the JSDO. If you call the method on `table-ref`, the method merges data only for the referenced temp-table. If you call the method on the JSDO itself, the method merges data for all temp-tables referenced for a ProDataSet.

**merge-object**

An object with the data to merge. If you call the method on `table-ref`, the object can either be an object that contains an array of record objects to merge with the referenced temp-table or a ProDataSet-formatted object containing such an array.

**Note:** This object must have a supported JavaScript object format that matches the data returned from the built-in read operation (JSDO fill( ) method). For example, the object returned from an invocation method for an output temp-table or ProDataSet that has the same schema as supported output from the built-in read operation should work.

The following formats are supported for `merge-object`:

- Single temp-table object with an array of record objects. For example:

  ```javascript
  {
    eCustomer: [
      // Record objects ...
    ]
  }
  ```

- An array of record objects for a single temp-table object or for a ProDataSet with a single temp-table object. For example:

  ```javascript
  [ // Record objects ...
  ]
  ```
addRecords( ) method

- A ProDataSet object with a single temp-table or multiple temp-table objects at the same level only. For example:

```javascript
{
    dsCustomerOrder: {
        eCustomer: [
            // Record objects ...
        ],
        eOrder: [
            // Record objects ...
        ]
    }
}
```

**Note:** If you pass a ProDataSet object with nested temp-table records, the method throws an error object.

**add-mode**

An integer that represents a merge mode to use. If you also specify `key-fields`, each merge mode handles duplicate keys in a particular manner as described here. If you **do not** specify `key-fields`, the method adds **all** the records of `merge-object` regardless of the mode. You can specify the following numeric constants, which affect how the temp-table record objects in `merge-object` are added to JSDO local storage:

- **progress.data.JSDO.MODE_APPEND** — Adds the temp-table record objects in `merge-object` to the existing record objects in JSDO local storage. If a duplicate key is found between a record object in `merge-object` and a record object in local storage, the method throws an error.

- **progress.data.JSDO.MODE_MERGE** — Adds the temp-table record objects in `merge-object` to the existing record objects in JSDO local storage. If duplicate keys are found between record objects in `merge-object` and record objects in local storage, the method ignores (does not add) the record objects with duplicate keys in `merge-object`.

- **progress.data.JSDO.MODE_REPLACE** — Adds the temp-table record objects in `merge-object` to the existing record objects in JSDO local storage. If duplicate keys are found between record objects in `merge-object` and record objects in local storage, the record objects with duplicate keys in local storage are replaced with the corresponding records in `merge-object`.

- **progress.data.JSDO.MODE_EMPTY** — Empties all temp-table record objects from JSDO local storage and replaces them with the contents of `merge-object`.

**Note:** If `merge-object` is an empty object (`{}`), this mode effectively empties the data from JSDO local storage.

**Caution:** If a temp-table `key-fields` that match the unique indices of corresponding temp-tables, adding the contents of `merge-object` can result in records with duplicate keys. If the corresponding temp-tables...
have unique indices, you must make any affected duplicate key fields unique before calling `saveChanges()`.

**key-fields**

An object with a list of key fields to check for records with duplicate keys. For example, when merging with a ProDataSet that has `eCustomer` and `eOrder` table references, you might use the following object:

```javascript
{
    eCustomer: [ "CustNum" ],
    eOrder: [ "CustNum", "Ordernum" ]
}
```

When merging with a single table reference, you might use the following array object:

```javascript
[ "CustNum", "Ordernum" ]
```

**Note:** For any `key-fields` that have the `String` data type, the character values for these fields are compared to identify duplicates according to the value of the `caseSensitive` property on each affected table reference.

**Note:** After this method checks for any duplicate keys and completes adding record objects to local storage from `merge-object`, and if you have set up automatic sorting using the `autoSort` property, all the record objects for the affected table references are sorted accordingly. If the sorting is done using sort fields, any `String` values in the specified sort fields are compared according to the value of the `caseSensitive` property.

A typical use for `addRecords()` is to merge additional data returned by an invocation method without having to re-load local storage with all the data from the `fill()` method.

For example, given a JSDO, `dataset`, that you fill with available records from the `eCustomer` and `eOrder` temp-tables, you might retrieve a new `eOrder` record as the result of a `getNewOrder()` invocation method on the JSDO and add the new record to JSDO local storage as follows:

```javascript
var dataset = progress.data.JSDO( "dsCustomerOrder" );
dataset.fill(); // Loads the ProDataSet with all available records

// Adds a new eOrder record retrieved from the service
var request = dataset.getNewOrder(null,false);
dataset.eOrder.addRecords( request.response, progress.data.JSDO.MODE_APPEND,
    [ "CustNum", "Ordernum" ],
    );
```

This code fragment adds the `eOrder` record for an existing `eCustomer` record specified by the `CustNum` property and a new order number specified by the `Ordernum` property of the single record object returned in `result.dsCustomerOrder.eOrder[0]`. 
addRecords( ) method

Note: OpenEdge adds each new record object with an OpenEdge-reserved Number field named _id, which uniquely identifies the record in JSDO local storage. Note that once you have saved the new record objects to the AppServer using saveChanges( ), their _id values can change with each invocation of the fill( ) method. For more information on these values, see the description of the getId( ) method.

Caution: Do not change the value referenced by _id. Otherwise, any Mobile App UI managed by OpenEdge can have unpredictable behavior.

See also: autoSort property, caseSensitive property, getId( ) method, fill( ) method, invocation method, saveChanges( ) method
afterCreate event

Fires after the JSDO, by means of a `saveChanges()` call following an `add()` call, sends a request to create a record and receives a response to this request from the AppServer.

**Applies to:** `progress.data.JSDO` class, `saveChanges()` method, table reference property (JSDO)

The following parameters appear in the signature of the event handler function:

**Syntax**

```
function (jsdo, record, success, request)
```

**jsdo**

A reference to the JSDO that invoked the create operation. For more information, see the description of the `jsdo` property of the `request` object.

**record**

A reference to the temp-table record upon which the create operation acted. For more information, see the description of the `jsrecord` property of the `request` object.

**success**

A Boolean that is `true` if the create operation was successful. For more information, see the description of the `success` property of the `request` object.

**request**

A reference to the request object returned after the create operation completes. For more information, see the description of the `request` object.
Example:

```javascript
/* subscribe to event */
myjsdo.subscribe('afterCreate', onAfterCreate);

/* some code that would add a record and save it */
var jsrecord = myjsdo.add();

myjsdo.saveChanges();

function onAfterCreate (jsdo , record , success , request ) {
    if (success) {
        /* for example, get the values from the record for redisplay */
        var myField = record.data.myField;
        // ...
    }
    else {
        if (request.response && request.response._errors &&
            request.response._errors.length > 0){
            var lenErrors = request.response._errors.length;
            for (var idxError=0; idxError < lenErrors; idxError++) {
                var errorEntry = request.response._errors[idxError];
                var errorNum = errorEntry._errorNum;
                var errorMsg = errorEntry._errorMsg;
                /* handle error */
            }
            /* handle error */
        }
    }
}
```

See also: add( ) method, record property, subscribe( ) method, unsubscribe( ) method
afterDelete event

Fires after the JSDO, by means of a saveChanges( ) call following a remove( ) call, sends a request to delete a record and receives a response to this request from the AppServer.

Applies to:  progress.data.JSDO class, saveChanges( ) method, table reference property (JSDO)

The following parameters appear in the signature of the event handler function:

Syntax

```javascript
function ( jsdo , record , success , request )
```

**jsdo**

A reference to the JSDO that invoked the delete operation. For more information, see the description of the jsdo property of the request object.

**record**

A reference to the temp-table record upon which the delete operation acted. For more information, see the description of the jsrecord property of the request object.

**success**

A Boolean that is true if the delete operation was successful. For more information, see the description of the success property of the request object.

**request**

A reference to the request object returned after the delete operation completes. For more information, see the description of the request object.
Example:

```javascript
/* subscribe to event */
myjsdo.subscribe('afterDelete', onAfterDelete);

/* some code that would delete a record and send to the server */
var jsrecord = myjsdo.findById(myid);
myjsdo.remove();
myjsdo.saveChanges();

function onAfterDelete (jsdo , record , success , request ) {
  if (success) {
    /* for example, get the values from the record that was
     deleted to display a confirmation message */
    var myKeyField = record.data.myKeyField;
    ...
  } else {
    if (request.response && request.response._errors &&
      request.response._errors.length > 0){
      var lenErrors = request.response._errors.length;
      for (var idxError=0; idxError < lenErrors; idxError++) {
        var errorEntry = request.response._errors[idxError];
        var errorMsg = errorEntry._errorMsg;
        var errorNum = errorEntry._errorNum;
        /* handle error */
      }
    } else {
    }
  }
};
```

See also: remove( ) method, subscribe( ) method, unsubscribe( ) method
afterFill event

Fires after the JSDO, by means of a fill( ) call, sends a request to read a temp-table or ProDataSet into JSDO local storage and receives a response to this request from the AppServer.

Applies to: progress.data.JSDO class, fill( ) method

The following parameters appear in the signature of the event handler function:

Syntax

```javascript
function ( jsdo , success , request )
```

**jsdo**

A reference to the JSDO that invoked the fill operation. For more information, see the description of the jsdo property of the request object.

**success**

A Boolean that is true if the fill operation was successful. For more information, see the description of the success property of the request object.

**request**

A reference to the request object returned after the fill operation completes. For more information, see the description of the request object.

Example:

```javascript
myjsdo.subscribe('afterFill', onAfterFill);
myjsdo.fill();

function onAfterFill(jsdo , success , request ) {
    if (success) {
        /* for example, add code to display all records on a list */
        jsdo.foreach(function (jsrecord) {
            /* you can reference the fields as jsrecord.data.field */
        });
    }
    else {
        if (request.response && request.response._errors &&
            request.response._errors.length > 0)(
            var lenErrors = request.response._errors.length;
            for (var idxError=0; idxError < lenErrors; idxError++) {
                var errorEntry = request.response._errors[idxError];
                var errorMsg = errorEntry._errorMsg;
                var errorNum = errorEntry._errorNum;
                /* handle error */
            }
        }
    }
};
```

See also: subscribe( ) method, unsubscribe( ) method
afterInvoke event

Fires after a non-built-in method is called asynchronously on a JSDO and a response to the request is received from the AppServer. Synchronous method calls do not cause this event to fire.

Appplies to: progress.data.JSDO class, invocation method

The following parameters appear in the signature of the event handler function:

Syntax

```javascript
function (jsdo, success, request)
```

*jsdo*

A reference to the JSDO that invoked the method. For more information, see the description of the jsdo property of the request object.

*success*

A Boolean that is true if the operation was successful. For more information, see the description of the success property of the request object.

*request*

A reference to the request object returned after the operation completes. For more information, see the description of the request object.

Example:

```javascript
myjsdo.subscribe('afterInvoke', 'myMethodName', onAfterInvokeMyMethodName);
myjsdo.myMethodName( paramObject);

function onAfterInvokeMyMethodName(jsdo, success, request) {
    if (success) {
        var response = request.result.response;
        var retval = response._retval;
        var myOutputParm = response.myOutParam;
    } else {
        if (request.response && request.response._errors &&
            request.response._errors.length > 0) {
            var lenErrors = request.response._errors.length;
            for (var idxError=0; idxError < lenErrors; idxError++) {
                var errorEntry = request.response._errors[idxError];
                var errorMsg = errorEntry._errorMsg;
                var errorNum = errorEntry._errorNum;
                /* handle error */
            }
        }
    }
}
```

See also: subscribe() method, unsubscribe() method
afterSaveChanges event

Fires once for each call to the `saveChanges()` method on a JSDO, after responses to all create, update, and delete requests have been received from the AppServer.

**Applies to:** `progress.data.JSDO` class, `saveChanges()` method

The following parameters appear in the signature of the event handler function:

**Syntax**

```javascript
function (jsdo, success, request)
```

**jsdo**

A reference to the JSDO that invoked the `saveChanges()` method. For more information, see the description of the `jsdo` property of the request object.

**success**

A Boolean that is true if all operations initiated by `saveChanges()` were successful. For more information, see the description of the `success` property of the request object.

**request**

A reference to the request object returned after all requested operations complete. For more information, see the description of the `request` object.
Example:

```javascript
/* subscribe to event */
myjsdo.subscribe('afterSaveChanges', onAfterSaveChanges);

/* some code that would do multiple CRUD operations and 
send them to the server */
var newrec = myjsdo.add();
...
var jsrecord = myjsdo.findById(myid);
myjsdo.remove();
myjsdo.saveChanges();

function onAfterSaveChanges(jsdo, success, request) {
    /* number of operations on batch */
    var len = request.batch.operations.length;

    if (success) {
        /* all operations in batch succeeded */
        /* for example, redisplay records in list */
        jsdo.foreach(function(jsrecord) {
            /* reference the record/field as jsrecord.data.fieldName */
        });
    } else {
        /* one or more operations in batch failed */
        for(var idx = 0; idx < len; idx++) {
            var operationEntry = request.batch.operations[idx];

            if (!operationEntry.success) {
                /* handle error condition */
                if (operationEntry.response && operationEntry.response._errors
                    && operationEntry.response._errors.length > 0){
                    var lenErrors = operationEntry.response._errors.length;
                    for (var idxError=0; idxError < lenErrors; idxError++) {
                        var errors = operation.response._errors[idxError];
                        var errorMsg = errors._errorMsg;
                        var errorNum = errors._errorNum;
                        /* handle error */
                    }
                } else {
                    /* operation succeeded */
                }
            }
        }
    }
}

See also: subscribe() method, unsubscribe() method
```
afterUpdate event

Fires after the JSDO, by means of a `saveChanges()` call following an `assign()` call, sends a request to update a record and receives a response to this request from the AppServer.

**Applies to:** `progress.data.JSDO` class, `saveChanges()` method, table reference property (JSDO)

The following parameters appear in the signature of the event handler function:

**Syntax**

```javascript
function (jsdo, record, success, request)
```

**jsdo**

A reference to the JSDO that invoked the update operation. For more information, see the description of the `jsdo` property of the request object.

**record**

A reference to the temp-table record upon which the update operation acted. For more information, see the description of the `jsrecord` property of the request object.

**success**

A Boolean that is true if the update operation was successful. For more information, see the description of the `success` property of the request object.

**request**

A reference to the request object returned after the update operation completes. For more information, see the description of the `request` object.
Example:

```javascript
/* subscribe to event */
myjsdo.subscribe('afterUpdate', onAfterUpdate);

/* some code that would update a record and send to the server */
var jsrecord = myjsdo.findById(myid);
myjsdo.assign( updatedDataObject );
myjsdo.saveChanges();

function onAfterUpdate (jsdo, record, success, request) {
    if (success) {
        /* for example, get the values updated by the server from the record to redisplay */
        var newValue = record.data.myField;
        ...
    } else {
        if (request.response && request.response._errors &&
            request.response._errors.length > 0){
            var lenErrors = request.response._errors.length;
            for (var idxError=0; idxError < lenErrors; idxError++) {
                var errorEntry = request.response._errors[idxError];
                var errorMsg = errorEntry._errorMsg;
                var errorNum = errorEntry._errorNum;
                /* handle error */
            }
        }
    }
}
```

See also: assign( ) method (JSDO class), subscribe( ) method, unsubscribe( ) method
assign( ) method (JSDO class)

Updates field values for the specified record object in JSDO local storage. The specified record object can be either the working record of a referenced temp-table or any record provided by a JSRecord object.

After execution, any working records previously set before the method executed remain as the working records. To synchronize the change on the AppServer, call the saveChanges( ) method.

Return type: Boolean

Applies to: JSRecord object, progress.data.JSDO class, table reference property (JSDO)

Syntax

```
[table-ref.] assign ( update-object )
jsrecord-ref.assign ( [] update-object ] )
```

**table-ref**

A table reference on the JSDO for a temp-table that has a working record. If the JSDO references only a single temp-table, the method can be called on the JSDO itself.

**jsrecord-ref**

A reference to a JSRecord object for a temp-table record in JSDO local storage.

**update-object**

Passes in the data to update the specified record object in JSDO local storage. Each property of the object has the name of a temp-table field name and the value to set for that field in the specified record. Any temp-table fields without corresponding properties in update-object remain unchanged in the record.

**Note:** After this method updates the specified record object, and if you have set up automatic sorting using the autoSort property, all the record objects for the affected table reference are sorted accordingly. If the sorting is done using sort fields, any String fields are compared according to the value of the caseSensitive property.

The following code fragment shows a jQuery event defined on a save button to save the current field values for a customer detail form to the corresponding eCustomer record in JSDO local storage:

```
dataSet = new progress.data.JSDO( 'dsCustomerOrder' );

$('#btnSave').bind('click', function(event) {
    var jsrecord = dataSet.eCustomer.findById($('#custdetail #id').val());
    jsrecord.assign(update-object);
    dataSet.saveChanges();
});
```
assign( ) method (JSDO class)

The form has been displayed with previous values of the same record. When the button is clicked, the event handler uses the findById( ) method to find the original record with the matching internal record ID (jsrecord) and invokes the assign( ) method on jsrecord with an object parameter to update the fields in eCustomer with any new values entered into the form.

See also: autoSort property, caseSensitive property, getSchema( ) method, saveChanges( ) method
assign( ) method (UIHelper class)

Updates the working record in JSDO local storage with the values currently displayed on the detail page form.

**Return type:** Boolean

**Applies to:** progress.ui.UIHelper class, table reference property (UIHelper)

**Syntax**

```
[table-ref.] assign ( )
```

**table-ref**

A table reference on the UIHelper instance. If the JSDO associated with the UIHelper instance references only a single temp-table, the method can be called on the UIHelper instance itself.

**Note:** After this method updates the working record in JSDO local storage, and if you have set up automatic sorting using the autoSort property, all the record objects for the affected table reference are sorted accordingly. If the sorting is done using sort fields, any String fields are compared according to the value of the caseSensitive property.

**See also:** autoSort property, caseSensitive property, getFormFields( ) method
async property

A Boolean that indicates, if set to true, that the Mobile operation was executed asynchronously on the Mobile App.

Data type: Boolean
Access: Read-only
Applies to: request object

This request object property is available only for the following events:

- afterCreate
- afterDelete
- afterFill
- afterInvoke

See also: add( ) method, remove( ) method, fill( ) method, invocation method
authenticationModel property

A string constant that specifies the type of authentication that the server requires from the Mobile App. Valid values are:

- **progress.data.Session.AUTH_TYPE_ANON** — No authentication is required. This is the default value.

- **progress.data.Session.AUTH_TYPE_BASIC** — The Mobile Web application requires a valid user ID and password, but does not provide a page containing a login form (credentials are typically entered in a generic login dialog provided by either the Mobile App, the browser, or the native device container in which the App is running). The Mobile Web application requires a valid user ID and password, but does not provide a page containing a login form (credentials are typically entered in a generic login dialog provided by either the Mobile App, the browser, or the native device container in which the App is running).

- **progress.data.Session.AUTH_TYPE_FORM** — The Mobile Web application requires a valid user ID and password and provides a page containing a login form.

Data type: String
Access: Writable
Applies to: progress.data.Session class

If the Mobile Web application requires authentication, you must set this value correctly to ensure that users can log in.

See also: login( ) method
autoSort property

A Boolean on a JSDO and its table references that indicates if record objects are sorted automatically on the affected table references in JSDO local storage at the completion of a supported JSDO operation. When set to true, and after you have specified a sorting method for each affected table reference, record objects are sorted after the JSDO operation completes its update of local storage. When set to false, or if no sorting method is specified for a given table reference, no automatic sorting occurs after the JSDO operation completes. The default setting is true for all table references of a JSDO.

Data type: Boolean
Access: Writable
Applies to: progress.data.JSDO class, table reference property (JSDO)

When set on a JSDO, the property setting affects the sorting of record objects for all table references in the JSDO. When set on a single table reference, the property setting affects the sorting of record objects only for the specified table reference. For example, to set this property to true on only a single table reference in the JSDO:

1. Set the value on the JSDO to false, which sets false on all its table references.
2. Set the value on the selected table reference to true, which sets true on only the this one table reference.

In order to activate automatic sorting for an affected table reference, you must invoke one of the following JSDO methods to specify a sorting method for the table reference:

- setSortFields( ) — Identifies the sort fields to use in the record objects and whether each field is sorted in ascending or descending order according to its data type. Any String fields specified for a table reference are sorted using letter case according to the setting of the caseSensitive property (false by default).

Note: Changing the value of the caseSensitive property triggers an automatic sort if the autoSort property is also set to true for the affected table reference.

- setSortFn( ) — Identifies a sort function that compares two record objects according to the criteria you specify and returns a value that indicates if one record sorts later than the other in the sort order, or if the two records sort at the same position in the sort order. The caseSensitive property setting has no effect on the operation of the specified sort function unless you choose to involve the setting of this property in your criteria for comparison.

If you specify both sort fields and a sort function to sort the record objects for a table reference, the sort function takes precedence. You can also call the setSortFields( ) and setSortFn( ) functions to clear one or both settings of the sort fields and sort function. However, at least one setting must be active for automatic sorting to occur on a table reference.

The following supported JSDO operations trigger automatic sorting on any affected table references before they complete their updates to JSDO local storage:
• **Invoking the `add()` method** — Sorts the record objects of the affected table reference.

• **Invoking the `addRecords()` method** — Sorts the record objects of either the single affected table reference or all affected table references in the JSDO. (Unaffected table references do not participate in the sort, including those for which `autoSort` is `false`, those for which no sort fields or sort function are set, or those other than the single JSDO table reference on which `addRecords()` is called, if it is called only on a single table reference.)

• **Invoking the `assign()` method (JSDO or UIHelper class)** — Sorts the record objects of the affected table reference.

• **Assigning a value to a field reference directly on the working record of a table reference** (`jsdo.table-ref.field-ref = value`) — Sorts the record objects of the affected table reference.

**Note:** Assignment to a field referenced on the `data` property *never* triggers automatic sorting (for example, `jsdo.table-ref.data.field-ref = value`)

• **Changing the value of the `caseSensitive` property** — Sorts the record objects of the affected table reference, or of all affected table references if the property value is changed on the JSDO.

• **Invoking the `fill()` method** — Sorts the record objects of all affected table references in the JSDO. (Unaffected table references do not participate in the sort, including any table references for which `autoSort` is `false`, or for which no sort fields or sort function are set.)

**Notes:** Invoking the `remove()` method does not trigger an automatic sort and has no effect on any existing sort order established for the table reference. However, if there is a sort order that depends on the presence or absence of the record object you are removing, and you want to establish the appropriate sort order when this record object is absent, you must manually sort the remaining record objects using the `sort()` method by passing it the same sort function that you used to establish the sort order when this record object was present.

Because automatic sorting executes in JavaScript on the client side, sorting a large set of record objects can take a significant amount of time and make the UI appear to be locked. You might set a wait or progress indicator just prior to any action that can sort a large record set to alert the user that the app is working.

In the following code fragment, automatic local sorting is turned off for all table references of the `dsCustOrds` JSDO by setting its `autoSort` property to `false`. Automatic sorting is then turned on for the `eCustomer` table reference of the JSDO by setting its `autoSort` value to `true` and using the `setSortFields()` method to set its `Name` field as the single, descending sort field:

---

**OpenEdge® Development: Mobile Applications** 237
When the `fill()` method executes on the JSDO, all the referenced temp-tables are loaded from the AppServer into JSDO local storage with their record objects already sorted in case-insensitive, primary key order (by default). The record objects for `eCustomer` are then sorted locally in case-insensitive, descending order of the `Name` field.

See also: `add()` method, `addRecords()` method, `assign()` method (JSDO class), `assign()` method (UIHelper class), `caseSensitive` property, `fill()` method, `setSortFields()` method, `setSortFn()` method, `sort()` method, `table reference` property (JSDO)
batch property

A reference to an object with a property named `operations`, which is an array containing the request objects for each of the several operations performed in response to a call to the JSDO `saveChanges()` method.

**Data type:** Object

**Access:** Read-only

**Applies to:** request object

This request object property is available only for the following events:

- `afterSaveChanges`
- `beforeSaveChanges`

**See also:** `saveChanges()` method
beforeCreate event

Fires before the JSDO, by means of a `saveChanges()` call following an `add()` call, sends a request the AppServer to create a record.

**Applies to:**  progress.data.JSDO class, `saveChanges()` method, table reference property (JSDO)

The following parameters appear in the signature of the event handler function:

**Syntax**

```javascript
function ( jsdo, record, request )
```

`jsdo`

A reference to the JSDO that is invoking the create operation. For more information, see the description of the `jsdo` property of the request object.

`record`

A reference to the temp-table record upon which the create operation is about to act. For more information, see the description of the `jsrecord` property of the request object.

`request`

A reference to the request object returned before the create operation begins. For more information, see the description of the request object.

**Example:**

```javascript
/* subscribe to event */
myjsdo.subscribe('beforeCreate', onBeforeCreate);

/* some code that would add a record and save it */
var jsrecord = myjsdo.add();
...
myjsdo.saveChanges();

function onBeforeCreate( jsdo, record, request ) {
  /* for instance, here you can update data in the record before it is sent to the server */
  record.data.myField = myvalue;
}
```

**See also:**  `add()` method, record property, `subscribe()` method, `unsubscribe()` method
beforeDelete event

Fires before the JSDO, by means of a `saveChanges()` call following a `remove()` call, sends a request the AppServer to delete a record.

**Applies to:** `progress.data.JSDO` class, `saveChanges()` method, table reference property (JSDO)

The following parameters appear in the signature of the event handler function:

**Syntax**

```javascript
function ( jsdo , record , request )
```

**jsdo**

A reference to the JSDO that is invoking the delete operation. For more information, see the description of the `jsdo` property of the request object.

**record**

A reference to the temp-table record upon which the delete operation is about to act. For more information, see the description of the `jsrecord` property of the request object.

**request**

A reference to the request object returned before the delete operation begins. For more information, see the description of the request object.

**Example:**

```javascript
/* subscribe to event */
myjsdo.subscribe('beforeDelete', onBeforeDelete);

/* some code that would delete a record and send to the server */
var jsrecord = myjsdo.findById(myid);
myjsdo.remove();
myjsdo.saveChanges();

function onBeforeDelete( jsdo , record , request ) {
    /* code to execute before sending request to the server */
}
```

**See also:** `remove()` method, `subscribe()` method, `unsubscribe()` method
beforeFill event

Fires before the JSDO, by means of a `fill()` call, sends a request to the AppServer to read a temp-table or ProDataSet into JSDO local storage.

**Applies to:** `progress.data.JSDO` class, `fill()` method

The following parameters appear in the signature of the event handler function:

**Syntax**

```javascript
function (jsdo, request)
```

**jsdo**

A reference to the JSDO that is invoking the read operation. For more information, see the description of the `jsdo` property of the `request` object.

**request**

A reference to the request object returned before the read operation begins. For more information, see the description of the `request` object.

**Example:**

```javascript
myjsdo.subscribe('beforeFill', onBeforeFill);
myjsdo.fill();

function osBeforeFill ( jsdo, request ) {
    /* for instance, do any preparation for receiving data from the server */
}
```

**See also:** `subscribe()` method, `unsubscribe()` method
beforeInvoke event

Fires when a non-built-in method is called asynchronously on a JSDO, before the request for the operation is sent to the AppServer.

**Applies to:**  progress.data.JSDO class, invocation method

The following parameters appear in the signature of the event handler function:

**Syntax**

```javascript
function (jsdo, request)
```

**jsdo**

A reference to the JSDO that is invoking the method. For more information, see the description of the jsdo property of the request object.

**request**

A reference to the request object returned before the operation begins. For more information, see the description of the request object.

**Example:**

```javascript
myjsdo.subscribe('beforeInvoke', 'myMethodName', onBeforeInvokeMyMethodName);
myjsdo.myMethodName( paramObject);

function onBeforeInvoke ( jsdo , request ) {
    /* code to execute before sending request to the server */
}
```

**See also:**  subscribe( ) method, unsubscribe( ) method
**beforeSaveChanges event**

Fires once for each call to the `saveChanges()` method on a JSDO, before any create, update, or delete requests are sent to the AppServer.

**Applies to:** `progress.data.JSDO` class, `saveChanges()` method

The following parameters appear in the signature of the event handler function:

**Syntax**

```javascript
function (jsdo, request)
```

**jsdo**

A reference to the JSDO that is invoking the `saveChanges` method. For more information, see the description of the `jsdo` property of the `request` object.

**request**

A reference to the request object returned before the requested save operations begin. For more information, see the description of the `request` object.

**Example:**

```javascript
myjsdo.subscribe('beforeSaveChanges', onBeforeSaveChanges);

/* some code that would do multiple CRUD operations and
 * send them to the server */
var newrec = myjsdo.add();
...
var jsrecord = myjsdo.findById(myid);
myjsdo.remove();
myjsdo.saveChanges();

function onBeforeSaveChanges (jsdo, request) {
  /* code to execute before sending request to the server */
};
```

**See also:** `subscribe()` method, `unsubscribe()` method
beforeUpdate event

Fires before the JSDO, by means of a `saveChanges()` call following an `assign()` call, sends a request the AppServer to update a record.

**Applies to:** progress.data.JSDO class, `saveChanges()` method, table reference property (JSDO)

The following parameters appear in the signature of the event handler function:

**Syntax**

```javascript
function ( jsdo, record, request )
```

- **jsdo**
  
  A reference to the JSDO that is invoking the update operation. For more information, see the description of the `jsdo` property of the `request` object.

- **record**
  
  A reference to the temp-table record upon which the update operation is about to act. For more information, see the description of the `jsrecord` property of the `request` object.

- **request**
  
  A reference to the request object returned before the update operation begins. For more information, see the description of the `request` object.

**Example:**

```javascript
myjsdo.subscribe('beforeInvoke', 'myMethodName',
  onBeforeInvokeMyMethodName);
myjsdo.myMethodName( paramObject );

function onBeforeInvoke ( jsdo, request ) {
  /* code to execute before sending request to the server */
}:
```

**See also:** `assign()` method (JSDO class), `subscribe()` method, `unsubscribe()` method
caseSensitive property

A Boolean on a JSDO and its table references that indicates if String field comparisons performed by supported JSDO operations are case sensitive or case-insensitive for the affected table references in JSDO local storage. When set to true, all supported comparisons on String fields for an affected table reference are case sensitive. When set to false, all supported comparisons on String fields for an affected table reference are case insensitive. The default setting is false for all table references of a JSDO. This default setting (case insensitive) matches the default setting for letter case comparison in OpenEdge ABL.

Data type: Boolean
Access: Writable
Applies to: progress.data.JSDO class, table reference property (JSDO)

When set on a JSDO, the property setting affects all table references in the JSDO. When set on a single table reference, the property setting affects only the specified table reference. For example, to set this property to true on only a single table reference in the JSDO:

1. Set the value on the JSDO to false, which sets false on all its table references.
2. Set the value on the selected table reference to true, which sets true on only the this one table reference.

The JSDO operations that follow this property setting in String field comparisons include:

• Sorting record objects in JSDO local storage, including automatic sorting using sort fields that you specify using the autoSort property and the setSortFields( ) method, and manual sorting using specified sort fields that you perform using the sort( ) method

  Note: Changing the value of this property triggers an automatic sort if the autoSort property is also set to true for the affected table reference.

• Merging record objects into JSDO local storage for all merge modes that perform record field comparisons during execution of the addRecords( ) method

Notes: Any default String field comparisons that you might do in JavaScript within the callback functions that you specify for other JSDO methods and events are always case sensitive according to JavaScript rules and ignore this property setting.

Unlike character string comparisons in ABL, all JSDO-supported String field comparisons include trailing spaces and ignore any OpenEdge-specified collation tables.

To conform to Unicode default letter case mapping, the JSDO support for case-insensitive String-field comparison and sorting relies on the toUpperCase( ) JavaScript function instead of the toLocaleUpperCase( ) JavaScript function. The latter function uses the locale letter case mapping, which might be different from the default letter case mapping in Unicode.
In the following code fragment, automatic local sorting is set up for the eCustomer table reference on the dsCustOrds JSDO, with its Name field as the single descending sort field. All other table references on dsCustOrds have no automatic local sorting set up by default. Because OpenEdge sorting on String fields is case-insensitive by default, the fragment makes the local sort on the Name field case sensitive by setting caseSensitive on eCustomer to true:

```javascript
dsCustOrds = new progress.data.JSDO( { name: 'dsCustomerOrders' });
dsCustOrds.autoSort = false.
dsCustOrds.eCustomer.autoSort = true.
dsCustOrds.eCustomer.setSortFields( "Name:descending" );
dsCustOrds.eCustomer.caseSensitive = true.
dsCustOrds.fill();
```

When the fill( ) method executes on the JSDO, after all the referenced temp-tables are loaded from the AppServer, with their record objects already sorted in case-insensitive, primary key order (by default), the record objects for eCustomer are then sorted locally in case-sensitive, descending order of the Name field.

See also: addRecords( ) method, autoSort property, setSortFields( ) method, sort( ) method
catalogURIs property

Returns the list of URIs used to load the JSDO catalogs to access the Mobile services provided by the Mobile Web application for which the current `Session` object manages a user login session.

**Data type:** String array  
**Access:** Read-only  
**Applies to:** `progress.data.Session` class

This list includes the URI for each JSDO catalog loaded using the `addCatalog()` method. To return a corresponding list of Mobile service names for which the JSDO catalogs are loaded, read the `serviceNames` property.

**See also:** `addCatalog()` method, `services` property
clearItems( ) method

Clears the items from a list view.

Return type: null

Applies to: progress.ui.UIHelper class, table reference property (UIHelper)

Syntax

```
[table-ref.]clearItems ( )
```

table-ref

A table reference on the UIHelper instance. If the JSDO associated with the UIHelper instance references only a single temp-table, the method can be called on the UIHelper instance itself.
clientContextId property

The value of the most recent client context identifier (CCID) that the Session object has found in the X-CLIENT-CONTEXT-ID HTTP header of a server response message. If none has yet been found, the value is null.

**Data type:** String  
**Access:** Read-only  
**Applies to:** progress.data.Session class

The Session object automatically detects, stores, and returns the CCID sent by any appropriately configured Mobile Web application for which it has started a login session. This CCID is the same as the value of the ClientContextId property on the ABL Progress.Lang.OERequestInfo class-based object that is passed from an AppServer client (in this case, the Mobile Web application) to the AppServer that is executing a Mobile request.

**Notes:** You can access this OERequestInfo object on the AppServer using the ABL CURRENT-REQUEST-INFO attribute of the AppServer SESSION system handle. This CCID value is also available as the ABL SESSION-ID attribute of the single sign-on (SSO) client-principal handle returned by the GetClientPrincipal( ) method of the same ABL OERequestInfo class-based object.

You can configure a Mobile Web application to send a CCID using the features for configuring any OpenEdge Web application to use SSO. For more information, see the sections on enabling SSO for a Web application in Chapter 6, “Deploying Mobile Applications.”

**See also:** login( ) method
**data property**

An object containing the data for a temp-table record associated with a JSRecord object.

**Data type:** Object  
**Access:** Read-only  
**Applies to:** JSRecord object

Each property (field-ref) of the object corresponds to a field (column) in the temp-table, where the property name is identical to a temp-table field name and the property value is the value for that field in the specified record.

**Caution:** Never write directly to a to a field-ref using this data property; in this case, use field-ref only to read the data. Writing to data using such a reference does not mark the record for update when calling the saveChanges( ) method, nor does it re-sort the record in local storage according to any order you have established using the autoSort property. To mark a record for update and automatically re-sort the record according to the autoSort property, you must assign a field value either by setting a jsdo.table-ref.field-ref for a working record or by calling the assign( ) method on a valid table-ref of JSRecord object reference. For information on a table-ref, see the reference entry on the table reference property (JSDO).

**See also:** add( ) method, assign( ) method (JSDO class), assign( ) method (UIHelper class), autoSort property, findById( ) method, foreach( ) method, table reference property (JSDO)
**display( ) method**

Copies the field values of a given record to corresponding fields in the current HTML document for display in the form on the detail page. The record is the working record of a temp-table referenced in the local storage of a JSDO that is associated with a UIHelper instance.

After execution, any working records previously set before the method executed remain as the working records.

**Return type:** null

**Applies to:** progress.ui.UIHelper class, table reference property (UIHelper)

**Syntax**

```
[tetable-ref.]display ( )
```

**table-ref**

A table reference on a UIHelper instance. If the JSDO associated with the UIHelper instance references only a single temp-table, the method can be called on the UIHelper instance itself. The table must have a working record.

If a form field's id attribute (or dsid attribute, if OpenEdge Mobile App Builder was used to design the form), as specified by the HTML DOM, matches the name of a record field, the form field displays the value of the record field. If no HTML field corresponds to a given record field, the value of that field is not displayed.

The following code fragment shows the `display( )` method displaying each record of an eCustomer temp-table in JSDO local storage to its respective row of a previously established jQuery listview:

```
DataSet = new progress.data.JSDO( 'dsCustomerOrder' );
uihelper = new progress.ui.UIHelper({ jsdo: dataSet });
uihelper.eCustomer.setListView({
    name: 'listview',
    format: '{CustNum}<br>{Name}<br>{State}',
    autoLink: true
});
```

**See also:** getFormFields( ) method
fill( ) method

Initializes the JSDO local storage with record objects from the data records in a single temp-table, or in one or more temp-tables of a ProDataSet, as returned by the built-in read operation of the Mobile resource for which the JSDO is created. This built-in operation is the single resource operation that is annotated in the Mobile interface with the "read" operation type. The result of calling this method replaces any prior data in JSDO local storage with the record objects returned by the built-in read operation.

After completing execution, the working record for each referenced temp-table is set to its first record, depending on any active parent-child relationships and automatic sort settings. So, for each child temp-table, the first record object is determined by its table reference sort order (if any) and its relationship to the related working record in its parent temp-table.

Return type:    null
Applies to:     progress.data.JSDO class

Syntax

```plaintext
fill ([ filter-string ])
```

filter-string

A string that can be used on the AppServer to select records to be returned, much like the WHERE option of the ABL record phrase. The actual format of this string and its affect on the records returned is determined by the ABL routine on the AppServer that uses it. For example, you might pass:

- A single key value (e.g., "30")
- A relational expression (e.g., "CustNum > 10 AND CustNum < 30")
- An actual WHERE string (e.g., 'Item.CatDescription CONTAINS "ski & (gog* ! pol*)"')

**Note:** The JSDO requires the URI for the "read" operation of the resource to contain the following query string: "?filter=~{filter~}", where filter is the name of a string input parameter defined for the ABL routine that implements the operation (INPUT filter AS CHARACTER).

**Caution:** Using an actual WHERE string for a dynamic ABL query can create a potential security issue.

If you do not specify filter-string, the records returned, again, depend on the ABL routine.

**Note:** After this method initializes JSDO local storage with record objects, and if you have set up automatic sorting using the autoSort property, the record objects of each affected table reference are sorted in local storage according to the sort order you have have established. If sorting is done using sort fields, any String fields are compared according to the value of the caseSensitive property on a given table reference. If the autoSort property setting is false...
**fill( ) method**

for a given table reference, its record objects are loaded in the order that they are serialized from the corresponding temp-table on the AppServer (by its primary key).

This method always executes asynchronously, and fires the following JSDO named events, shown in operational order:

1. **beforeFill event**
2. **afterFill event**

After this method completes execution, you can read the record objects of JSDO local storage by using the `find( )`, `findById( )`, `foreach( )`, and `getData( )` methods of the JSDO. You can return the schema for this data by using the `getSchema( )` method. You can create a new record object in local storage using the JSDO `add( )` method, and you can update or delete a single record object in local storage by using the `assign( )` or `remove( )` method, respectively. You can display a record in a form by calling the `display( )` method on a `UIHelper` instance. You can merge data returned by an invocation method with the data in local storage using the `addRecords( )` method.

The following code fragment shows the `fill( )` method invoked on a JSDO for a `ProDataSet` resource (**dsCustomerOrder**):

```javascript
dataset = new progress.data.JSDO('dsCustomerOrder');
dataset.fill();
```

**Note:** OpenEdge initializes every record object with an OpenEdge-reserved `Number` field named `_id`, which uniquely identifies each record in JSDO local storage. This field has no relationship to the internal `RECID` and `ROWID` values maintained for the records of an OpenEdge database. Instead, it is used to map data to any OpenEdge-managed UI that you create in a Mobile App. Note that the value assigned to `_id` for any given record object can change with each invocation of the `fill( )` method. For more information, see the description of the `getId( )` method.

**Caution:** Do not change the value referenced by `_id`. Otherwise, any Mobile App UI managed by OpenEdge can have unpredictable behavior.

**See also:** autoSort property, caseSensitive property, `getId( )` method, invocation method, `saveChanges( )` method, progress.ui.UIHelper class
find( ) method

Searches for a record in a temp-table referenced in JSDO local storage and returns a reference to that record if found. If no record is found, it returns `null`.

After completing execution, any record found becomes the working record for the associated temp-table. If the searched temp-table has child temp-tables, and the `useRelationships` property is `true`, the working record of the result set for each child is set to the first record as determined by the relationship to its respective parent. If a record is not found, the working record is not set, and the working records of any child temp-tables are also not set.

Return type: JSRecord object

Applies to: progress.data.JSDO class, table reference property (JSDO)

Syntax

```
[table-ref].find ( funcRef )
```

**table-ref**

A table reference on the JSDO. If the JSDO references only a single temp-table, the method can be called on the JSDO itself.

**funcRef**

A reference to a JavaScript function that returns a `Boolean` value and has the following signature:

Syntax

```
function [ func-name ] ( jsrecord-ref )
```

Where `func-name` is the name of a function that you define external to the `find( )` parameter list and `jsrecord-ref` is a JSRecord reference to the next available record on `table-ref`. You can then pass `func-name` to the `find( )` method as the `funcRef` parameter. Alternatively, you can specify `funcRef` as the entire inline function definition without `func-name`.

The `find( )` function executes your `funcRef` for each record of `table-ref`, until it returns `true`, indicating that `funcRef` has found the record. You can test the field values on the `data` property of `jsrecord-ref` to determine the result. Otherwise, you return `false`, and the `find( )` function executes `funcRef` for the next available record.

If `funcRef` finds the record, `find( )` completes execution with both its return value and the `record` property of `table-ref` set to the JSRecord reference of the found working record. If `find( )` reaches the end of available records without `funcRef` returning `true`, `find( )` completes execution with both its return value and the `record` property on `table-ref` set to `null`, indicating that the sought for record is not found.

If `table-ref` references a child temp-table in a ProDataSet, when the `useRelationships` property is `true`, `find( )` uses the relationship to filter out all but the child records of the working record in the parent temp-table. However, if the working
record of the parent is not set, \texttt{find}() throws an error. If \texttt{useRelationships} is \texttt{false}, the search includes all records of the child temp-table and no error is thrown.

In following code fragment, \texttt{jsdo} references a single \texttt{customer} temp-table:

```javascript
var jsdo = new progress.data.JSDO( 'customer' );

jsdo.find(function(jsrecord) {
    return (jsrecord.data.CustNum == 10);
});
```

The inline function passed to \texttt{find()} returns \texttt{true} or \texttt{false} based on the value of the \texttt{CustNum} property of the object returned by the \texttt{data} property for the currently available \texttt{JSRecord} reference.

\textbf{See also:} \texttt{data} property, \texttt{foreach()} method, \texttt{record} property
findById( ) method

Locates and returns the record in JSDO local storage with the internal ID you specify. If no record is found, it returns null. You can access the internal ID of a record by calling the getId( ) method.

After completing execution, any record found becomes the working record for the associated temp-table. If the searched temp-table has child temp-tables, and the useRelationships property is true, the working record of the result set for each child is set to the first record as determined by the relationship to its respective parent. If a record is not found, the working record is not set, and the working records of any child temp-tables are also not set.

Return type:  JSRecord object
Applies to:  progress.data.JSDO class, table reference property (JSDO)

Syntax

```javascript
[table-ref.]findById ( id )
```

table-ref

A table reference on the JSDO for a temp-table that has a working record. If the JSDO references only a single temp-table, the method can be called on the JSDO itself.

id

The internal record ID used to match a record of table-ref. This is the same value originally returned for the record using the getId( ) function. It is typically used to create a jQuery listview row to display the record or a detail form used to display the record in the current HTML document. Later, when a listview row or detail form is selected, the corresponding id attribute with this value can be used to return the record from the JSDO, possibly to update the record with new data values input by the user.

If findById( ) locates a record with the matching record ID, it completes execution with both its return value and the record property of table-ref set to the JSRecord reference of the found working record. If the function does not locate the record, it completes execution with both its return value and the record property on table-ref set to null, indicating that no record of table-ref has a matching internal record ID.

If table-ref references a child temp-table in a ProDataSet, when the useRelationships property is true, findById( ) uses the relationship to filter out all but the child records of the working record in the parent temp-table; the remaining child records are excluded from the search. If useRelationships is false or the working record of the parent is not set, the search includes all records of the child temp-table and no error is thrown.
The following code fragment shows a jQuery event defined on a save button to save the current field values for a customer detail form to the corresponding eCustomer record in JSDO local storage:

```javascript
dataSet = new progress.data.JSDO('dsCustomerOrder');
$('#btnSave').bind('click', function(event) {
    var jsrecord = dataSet.eCustomer.findById($('#custdetail #id').val());
    jsrecord.assign();
    dataSet.saveChanges();
});
```

The form has been displayed with previous values of the same record. When the button is clicked, the event handler finds the original eCustomer record by calling
findById( ) with the id attribute of the form $('#custdetail #id').val(), which is set to the internal ID of the record. The jsrecord.assign( ) method then updates the record from the values of the corresponding form fields and saveChanges( ) invokes the resource "update" operation on the AppServer to save the updated record to its data source.

See also: data property, foreach( ) method, getId( ) method
fnName property

For an invoke operation, the name of the JSDO invocation method that called the operation. The \texttt{fnName} property is null in the case of a request object returned by a built-in create, read, update, or delete method.

| Data type: | String |
| Access:    | Read-only |
| Applies to: | request object |

This request object property is available only for the following event:

- afterInvoke

\textbf{Note:} The value of the \texttt{fnName} property is the same as that of the \texttt{op-name} parameter passed to the \texttt{subscribe( )} method that subscribed to the current invoke operation event.

\textbf{See also:} invocation method, \texttt{subscribe( )} method
foreach( ) method

Loops through the records of a temp-table referenced in JSDO local storage and invokes a function as a parameter on each iteration. With each iteration, it also sets the current record as the working record and passes it as a parameter to the function. This function can then operate on the working record and return a value indicating whether the `foreach( )` terminates the loop or invokes the function for the next working record of the temp-table.

If the referenced temp-table has child temp-tables, and the `useRelationships` property is `true`, with each iteration through the loop, the working record of the result set for each child is set to the first record as determined by the relationship to its respective parent.

After completing execution, the working records of the associated temp-table, and any child temp-tables, are the most recent working records established when the function terminates the loop.

Return type: null

Applies to: progress.data.JSDO class, table reference property (JSDO)

Syntax

```
[ table-ref. ] foreach ( funcRef )
```

table-ref

A table reference on the JSDO for a temp-table that has a working record. If the JSDO references only a single temp-table, the method can be called on the JSDO itself.

funcRef

A reference to a JavaScript function that returns a Boolean value and has the following signature:

Syntax

```
function [ func-name ] ( jsrecord-ref )
```

Where `func-name` is the name of a function that you define external to the `foreach( )` parameter list and `jsrecord-ref` is a JSRecord object reference to the next working record on `table-ref`. You can then pass `func-name` to the `foreach( )` method as the `funcRef` parameter. Alternatively, you can specify `funcRef` as the entire inline function definition without `func-name`.

The `foreach( )` function executes your `funcRef` for each record of `table-ref`, making this record the working record and passing it in as `jsrecord-ref`. You can then access the field values of the working record using the `data` property on `jsrecord-ref` or any field references available from `table-ref`. You can also invoke other JSDO methods, for example, to operate on the working record, including additional calls to `foreach( )` to operate on working records of any child temp-tables.
Your `funcRef` can terminate the `foreach()` loop be returning `false`. If the function does not return `false`, the loop continues.

If `table-ref` references a child temp-table in a `ProDataSet`, when the `useRelationships` property is true, `foreach()` uses the relationship to filter out all but the child records of the working record in the parent temp-table. However, if the working record of the parent is not set, `foreach()` throws an error. If `useRelationships` is false, the loop includes all records of the child temp-table and no error is thrown.

After creating a `JSDO` for a `dsCustomer` resource and loading it with record objects, the following code fragment shows the `foreach()` method looping through `eCustomer` records in `JSDO` local storage and displaying the `CustNum` and `Name` fields from each record, one record per line, to the current HTML page, and also to the console log:

```javascript
jsdo = new progress.data.JSDO({ name: 'dsCustomer' });
jsdo.subscribe('AfterFill', onAfterFillCustomers, this);
jsdo.fill();

function onAfterFillCustomers(jsdo, success, request) {
  jsdo.eCustomer.foreach(function(customer) {
    document.write(customer.data.CustNum + ' ' + customer.data.Name + '<br>);
    console.log(customer.data.CustNum + ' ' + customer.data.Name);
  });
}
```

See also: `data property`, `find( ) method`, `JSRecord object`
getData( ) method

Returns an array of record objects for a temp-table referenced in JSDO local storage. If this is a child temp-table, and the useRelationships property is true, the specific record objects in the result set depends on the relationship to its parent.

After completing execution, any working records previously set before the method executed remain as the working records.

**Return type:** Object array

**Applies to:** progress.data.JSDO class, table reference property (JSDO)

**Syntax**

```
[<table-ref>.]getData ( )
```

**table-ref**

A table reference on the JSDO. If the JSDO references only a single temp-table, the method can be called on the JSDO itself.

**See also:** getSchema( ) method
getFormFields( ) method

Reads the schema of the specified table and returns HTML text that defines corresponding input fields to appear on the form on the detail page. The returned HTML includes a hidden field for the internal record ID.

Return type: String
Applies to: progress.ui.UIHelper class, table reference property (UIHelper)

Syntax

```
[table-ref].getFormFields([ array ])
```

**table-ref**

A table reference on the UIHelper instance. If the JSDO associated with the UIHelper instance references only a single temp-table, the method can be called on the UIHelper instance itself.

**array**

An array of strings specifying a subset of fields to be returned from the table schema. If this parameter is omitted, all fields are returned. For example:

```
uiHelper.eCustomer.getFormFields([ '_id', 'CustNum', 'Name' ]);  
```

OpenEdge Mobile provides a default form field template for a jQuery Mobile environment. The following code fragment shows the default template values:

```
<div data-role="fieldcontain">
  <label for="{__name__}">{__label__}</label>
  <input id="{__name__}" name="{__name__}" placeholder="" value="" type="text" />
</div>
```

As shown above, the default template uses the following substitution parameters:

- `{__name__}` — The field name as defined in the schema
- `{__label__}` — The field's title property as defined in the schema

You can define a template to be used in place of the default by calling `setFieldTemplate( )`.

See also: `setFieldTemplate( ) method`
getFormRecord( ) method

Returns a JSRecord object for the record shown on the form on the detail page.

**Return type:** String

**Applies to:** progress.ui.UIManager class, table reference property (UIHelper)

**Syntax**

```
[table-ref.]getFormRecord { [array] }
```

**table-ref**

A table reference on the UIManager instance. If the JSDO associated with the
UIHelper instance references only a single temp-table, the method can be called
on the UIManager instance itself.
getId( ) method

Returns the unique internal ID for the record object referenced in JSDO local storage. The specified record object can be either the working record for a referenced temp-table, or any record provided by a JSRecord object.

After execution, any working records previously set before the method executed remain as the working records.

Return type: String

Applies to: JSRecord object, progress.data.JSDO class, table reference property (JSDO)

Syntax

```
[table-ref].getId ( )
jsrecord-ref.getId ( )
```

table-ref

A table reference on the JSDO for a temp-table that has a working record. If the JSDO references only a single temp-table, the method can be called on the JSDO itself.

jsrecord-ref

A reference to a JSRecord object for a temp-table record in JSDO local storage.

The internal record ID returned by this function is a unique value generated by OpenEdge for each record object loaded in JSDO local storage using the fill( ), add( ), or addRecords( ) methods. OpenEdge stores this value for each record object in a field referenced with the reserved name, _id. OpenEdge uses this record ID in the client JSDO Services built into the Mobile App Builder, and also by the progress.ui.UIManager class, to map the record objects in JSDO local storage to the HTML elements of Mobile Apps. You can pass any _id value returned by this method to the findById( ) method called on the associated temp-table reference to set the specified record as the working record for the table reference.

Note: The value assigned to _id for any given record object can change with each invocation of the fill( ) method.

Caution: Do not change the value referenced by _id. Otherwise, any Mobile App UI managed by OpenEdge can have unpredictable behavior.

See also: add( ) method, addRecords( ) method, fill( ) method, findById( ) method, foreach( ) method
getListViewRecord( ) method

Returns a JSRecord object for a specified item in a list view.

Return type:     JSRecord
Applies to:      progress.ui.UIManager class, table reference property (UIHelper)

Syntax

```javascript
[view-region].getListViewRecord ( list_item )
```

table-ref

A table reference on the UIManager instance. If the JSDO associated with the
UIManager instance references only a single temp-table, the method can be called
on the UIManager instance itself.

list_item

The HTML list item element corresponding to the desired record.

See also:    setListView( ) method, addItem( ) method
getSchema( ) method

Returns an array of objects, one for each field that defines the schema of a temp-table referenced in JSDO local storage. The properties of each object define the schema elements of the respective field.

After completing execution, any working records previously set before the method executed remain as the working records.

**Return type:** Object array

**Applies to:** progress.data.JSDO class, table reference property (JSDO)

**Syntax**

```
[table-ref.]getSchema ( )
```

**table-ref**

A table reference on the JSDO. If the JSDO references only a single temp-table, the method can be called on the JSDO itself.

**See also:** getData( ) method, fill( ) method
Any method on the JSDO that is defined by the Mobile resource to execute a corresponding ABL routine on the AppServer as an invoke operation. This can be any ABL routine in the Mobile interface that is annotated with an "invoke" operation type. The invocation method name can be the same as the ABL routine or an alias, as defined by the resource. The method passes any ABL input parameters as properties of an object parameter. The method returns results from the ABL routine, including any return value and output parameters, as properties of a request object that is returned by the method.

**Note:** The results of an invoke operation have no effect on JSDO local storage.

After completing execution, any working records previously set before the method executed remain as the working records.

**Return type:** request object

**Applies to:** progress.data.JSDO class

**Syntax**

```
op-name ( [ input-object [ , async-flag ] ] )
```

*op-name*

The name (specified as an identifier) of the invocation method as defined by the Mobile resource.

*input-object*

An object whose properties and values match the case-sensitive names and data types of the input parameters specified for the ABL routine. If the routine does not take input parameters, specify null or leave out the argument entirely.

*async-flag*

A Boolean that when true causes the method to execute asynchronously and when false causes the method to execute synchronously. The default value is true.

For a synchronous invocation, the method returns a request object that contains several properties depending on the status of the invoke operation. However, if there are any ABL output parameters or return value, they are returned as properties of an object referenced by the response property of the request object. The response object properties for output parameters match the case-sensitive names and data types of the output parameters specified for the ABL routine. Any return type is returned by an OpenEdge-defined _retVal property with a matching data type.

For an asynchronous invocation, the method returns a similar request object as input to any event handler function subscribed to the following named events that fire in the following operational order:

1. beforeInvoke event
2. afterInvoke event
**Note:** If you are calling an invocation method that either sends a temp-table or ProDataSet object as a property of `input-object` or returns a temp-table or ProDataSet object as a property of the `response` property object, you need to apply a rule in order to access this temp-table or ProDataSet object. The rule is that wherever you dereference or reference a temp-table or ProDataSet object, you must reference that value twice, separated by a period or a colon, depending on the context. For example, to access a temp-table object, `ttCust` returned by the `response` property in a `request` object, you must code the following dereference: `request.response.ttCust.ttCust`. Similarly, if you pass `ttCust` to an invocation method, `InputTT()`, you must code the following reference: `jado.InputTT( {ttCust: {ttCust:ttCust}});`

**See also:** `fill()` method, record property, request object, `saveChanges()` method
jsdo property

An object reference to the JSDO that performed the operation returning the request object.

Data type: progress.data.JSDO class
Access: Read-only
Applies to: request object

This request object property is available for all events.

See also: fill( ) method, invocation method, saveChanges( ) method
jsrecord property

An object reference to the record created, updated, or deleted by the operation.

Data type: JSRecord object
Access: Read-only
Applies to: request object

This request object property is available only for the following events:

- afterCreate
- afterDelete
- afterUpdate
- beforeCreate
- beforeDelete
- beforeUpdate

See also: saveChanges( ) method
lastSessionXHR property

Returns an object reference to the XMLHttpRequest object (XHR) that was most recently used by the progress.data.Session object to execute a Session object method. The one exception is in the case of a successful invocation of the logout( ) method, in which case lastSessionXHR is set to null.

Note: This does not include the XMLHttpRequest objects that a Session object helps to prepare for JSDO AppServer requests (Mobile operations).

Data type: Object
Access: Read-only
Applies to: progress.data.Session class

It is possible for a Session object method to fail prior to sending a request to the Web server. This is especially true of the addCatalog( ) method. If a Session object method fails before sending the request, lastSessionXHR is the most recent XHR returned from a previous method call that did send a request to the Web server. For example, if login( ) fails with an authentication error on the Web server, and the application follows with a call to addCatalog( ), addCatalog( ) throws an error. However, lastSessionXHR then returns the XHR used for the unsuccessful login( ) request instead of for the failed addCatalog( ) call, because addCatalog( ) never attempts to send its own request and therefore doesn’t create an XHR.

In general, you can use the XHR returned by this property to find out more information about the results of a Web server request than you can identify from the error code returned, or the error object thrown, by a given Session object method. One possible scenario is that the request to the server can succeed, but the body of the response, which should contain the catalog, contains data that cannot be parsed successfully as a JSDO catalog. For example, the following code might provide more information about what has gone wrong:

```javascript
// create Session
pdsession = new progress.data.Session();

// log in
pdsession.login('/SportsApp');

// load catalog
try {
    pdsession.addCatalog("/SportsApp/static/mobile/CustomerSvc.json");
} catch(e) {
    var xhr = pdsession.lastSessionXHR;
    if (xhr) {
        if (xhr.status === 200) { // did HTTP request succeed?
            // probably something wrong with the catalog, dump it
            if (xhr.responseText) {
                console.log(xhr.responseText);
            }
        }
    }
}
```

See also: addCatalog( ) method, login( ) method, logout( ) method
login( ) method

Starts a user login session on the current Session object by sending an HTTP request with user credentials to a URI for a specified Mobile Web application. If the user credentials are authenticated, the login session is started and assigned a unique session ID for that session that is also passed to the AppServer that implements the Mobile Web application. If the Mobile Web application is so configured, along with the session ID, it also sends a single sign-on (SSO) client-principal with the user credentials to the application AppServer. The method also returns a code indicating the success of the login request.

Notes: Before invoking this method, ensure that you set the authenticationModel property on the Session object correctly (see the notes on authentication models).

If the browser or mobile device has already authenticated a user login session, this method completes successfully.

This method does not support proxy servers (servers that function as a security service) in Release 11.2.

Return type: Number
Applies to: progress.data.Session class

Syntax

```javascript
login ( service-uri [, username , password [, login-target ]])
```

**service-uri**

A string expression containing the URI of the Mobile Web application for which to start the user login session. This Mobile Web application must support one or more OpenEdge Mobile services in order to create JSDOs for the service resources provided by the application. If HTTP Basic Authentication is in effect for the Mobile Web application (see the notes on authentication models), this URI is appended with a string that identifies a protected resource against which to authenticate the login session (see login-target).

If the Mobile App from which you are logging in is a Mobile Web App deployed to the same Apache Tomcat server as the Mobile Web application, you can specify service-uri as a relative URI, for example, /SportsMobileApp, which is relative to the deployment end point (Tomcat server domain or host and port).

If the Mobile App from which you are logging in is a Mobile Native App that will be installed to run directly in a native device container, or if it is a Mobile Web App deployed to a different Web server from the Mobile Web application, you must specify service-uri as an absolute URI to the Tomcat server domain or host and port, for example, http://www.progress.com/SportsMobileApp, or perhaps for testing, http://testmach:8980/SportsMobileApp.

Note: For any Mobile App that must specify service-uri as an absolute URI, you must maintain separate JavaScript sources to deploy the Mobile App
The login( ) method for different Mobile Web application environments, such as one for testing and one for production.

**username**

A string expression containing a user ID for the method to send to the Web server for authentication.

**Note:** The **username** property returns the most recent value passed to this method for the current **Session** object.

**password**

A string expression containing a password for the method to send to the Web server to authenticate the specified user.

**login-target**

A string expression that when appended to **service-uri** specifies a Mobile Web application resource against which the specified user is authenticated. If you do not specify a value for **login-target**, the value is set to "/static/home.html" by default.

**Note:** The value returned by the **loginTarget** property of the **Session** object is either the value of the **login-target** parameter or the default ("/static/home.html").

When the method completes, it returns one of the following numeric constants to indicate the result:

- **progress.data.Session.SUCCESS** — User login session started successfully.
- **progress.data.Session.AUTHENTICATION_FAILURE** — User login failed because of invalid user credentials (**username** or **password**).
- **progress.data.Session.GENERAL_FAILURE** — User login failed because of a non-authentication failure.

You can also return the result for the most recent login attempt on the current **Session** object by reading the **loginResult** property. For a more specific status code returned in the HTTP response, you can check the value of the **loginHttpStatus** property. For more detailed information about any response (successful or unsuccessful) returned from the Web server, you can also check the XMLHttpRequest object (XHR) returned by the **lastSessionXHR** property.

The general Web server interaction with and response to this method depends on the authentication model that the Web server uses and how resources are accessed and protected. You configure the authentication model for each Mobile Web application deployed to the Apache Tomcat.
OpenEdge Mobile supports the following authentication models:

- **Anonymous** — No authentication is required. This is the default value.

- **HTTP Basic Authentication** — The Mobile Web application requires a valid user ID and password, but does not provide a page containing a login form (credentials are typically entered in a generic login dialog provided by either the Mobile App, the browser, or the native device container in which the App is running).

- **HTTP Forms Authentication** — The Mobile Web application requires a valid user ID and password and provides a page containing a login form.

For more information on these authentication models and how to configure them for a Mobile Web application, see the sections on Web server authentication models in Chapter 6, “Deploying Mobile Applications.” For more information on the interaction between this method and the Web server, see the sections on managing login session in Chapter 4, “Creating Mobile Apps using JSDOs.”

**Caution:** You must be sure that security is configured to complete authentication before the application requests resources in the JSDO catalog. Although it is possible to configure application security so that the only the Mobile resources in the catalog require authentication, Progress Software does not recommend this approach. Instead, Progress Software recommends that you require authentication for application resources in addition to those defined in the catalog, and require that the authentication occur prior to accessing any resources in the catalog. *(Note: This is the purpose of the login-target parameter, either one you pass to the login( ) method or its default.)* Once the user is authenticated, the Web server provides access to all other resources, including catalog resources, according to the user's authorization settings.

**Note:** Unless the application design guarantees that the user will be prompted by the Web browser or native device container to provide credentials before a login( ) call occurs, Progress Software recommends (in some cases requires) that the Mobile App pass the credentials as parameters to the login( ) method. In addition, you must correctly set the value of the Session object’s authenticationModel property. Coding the Mobile App in this way ensures that the proper credentials are submitted to the server and promotes a favorable user experience.

**See also:** addCatalog( ) method, lastSessionXHR property, loginHttpStatus property, loginResult property, loginTarget property, logout( ) method, serviceURI property, userName property, authenticationModel property
loginHttpStatus property

Returns the specific HTTP status code returned in the response from the most recent login attempt on the current Session object.

Data type: Number
Access: Read-only
Applies to: progress.data.Session class
See also: login( ) method
**loginResult property**

Returns the return value of the `login()` method, which is the basic result code for the most recent login attempt on the current `Session` object.

**Data type:** Number  
**Access:** Read-only  
**Applies to:** `progress.data.Session` class

Possible login return values include the following numeric constant values:

- `progress.data.Session.LOGIN_SUCCESS` — User login session started successfully.
- `progress.data.Session.LOGIN_AUTHENTICATION_FAILURE` — User login failed because of invalid user credentials.

For a more specific status code returned in the HTTP response, you can check the value of the `loginHttpStatus` property.

**See also:** `login()` method, `loginHttpStatus` property
loginTarget property

Returns the string appended to the Mobile Web application URI passed to the `login()` method to form the URI of an application resource against which the user has been authenticated for the current login session. By default, this appended string is `/static/home.html`.

**Data type:** String  
**Access:** Read-only  
**Applies to:** `progress.data.Session` class

You initially provide the Mobile Web application URI as a parameter to the `login()` method. You can also pass a parameter to this method to specify a non-default value for the string appended to this URI.

**See also:** `login()` method, `serviceURI` property
logout( ) method

Terminates the login session on the Web server associated with the Session object, and invalidates any session currently maintained by the server and the browser or hybrid native wrapper. Once logout( ) is executed, no further communication (other than a login( ) call) can occur between the Mobile App and the server until a new login session is established.

Return type: null
Applies to: progress.data.Session class

Syntax

```
logout ( )
```

When this method terminates the associated login session, the Session object can be re-used to start a new session. The Session object’s properties retain their values, with the following exceptions:

- loginResult is reset to null.
- loginHttpStatus is reset to null.
- clientContextId is reset to null.

Existing JSDOs and catalog information are not affected by logout( ). However, any attempt to call addCatalog( ) or a JSDO method that requires contacting the server results in an error being thrown.

For detailed information about any unsuccessful logout( ) response returned from the Web server, you can check the XMLHttpRequest object (XHR) returned by the lastSessionXHR property. However, note that if the logout( ) call is successful, lastSessionXHR is set to null.

Note: If the Web server is using the HTTP Basic Authentication model, you do not need to invoke this method unless you want to re-use the Session object.

See also: lastSessionXHR property, login( ) method
**name property**

The name of the Mobile resource for which the current JSDO is created. This value must match the name of a resource provided by the Mobile service for which a login session has already been started.

**Data type:** String  
**Access:** Writable  
**Applies to:** progress.data.JSDO class

**Note:** To set this property, you must pass its value to the JSDO constructor.

**See also:** services property, progress.data.Session class
onOpenRequest property

Returns a developer-specified callback function that the Session object executes to modify a request object before sending the request object to the server. Such a function might serve, for example, to add a header to the request.

It is not normally necessary to use this property, because OpenEdge Mobile properly handles preparation of the request object for normal circumstances.

Data type: function  
Access: Writable  
Applies to: progress.data.Session class

By default, the value of the onOpenRequest property is null, meaning that the request object is sent without modification. If the value is set to a function, the function takes a single parameter.

Syntax

```javascript
mysession.onOpenRequest = myFunction( params )
```

params

An object that has the following properties:

- **xhr** — An object reference to the XMLHttpRequest object (XHR) to be used to send the request. The current request object can be modified by the function. When the callback is called, XMLHttpRequest.open( ) will already have been called on the XHR, but the callback can call open( ) again, overriding the effects of the first open( ). When the callback function is used for a login( ), addCatalog( ), or logout( ) call, although it should not be necessary and is not recommended, it is possible to replace the XHR entirely by creating a new object and assigning it as the value of the xhr property.

- **verb** — The HTTP operation (GET, PUT, etc.) to be performed by the request.

- **uri** — The URI to which the request is addressed.

- **session** — A reference to the Session object that invoked the callback.

- **formPreTest** — A Boolean specifying whether the current login( ) request is a preliminary request, used in cases of Form authentication, to determine whether the user is already logged in (true) or an actual login request (false).

- **async** — A Boolean specifying whether the request is asynchronous (true) or synchronous (false).

**Note:** If the callback function is used for a login( ), addCatalog( ), or logout( ) call, and if it calls XMLHttpRequest.open( ), the request must be sent synchronously.
If you assign a function as the value of `onOpenRequest`, it remains in effect for all requests for the duration of the session unless it is replaced by another function or is set to `null`. Therefore, be sure to reset the value of the property as necessary, as in the following example:

```javascript
mysession.onOpenRequest = function( params ) {
    params.xhr.setRequestHeader('Authorization', auth);
};
mysession.login(serviceURI, username, password);
mysession.onOpenRequest = null;
```

See also: request object, xhr property
**paramObj property**

A reference to the object, if any, that was passed as a parameter to the method that returned the request object. If no parameter was passed to the method, the `paramObj` property is undefined.

- **Data type:** Object
- **Access:** Read-only
- **Applies to:** request object

This request object property is available for all events where it is passed, but does not apply to the following events:

- afterSaveChanges
- beforeSaveChanges

**See also:** fill( ) method, invocation method, saveChanges( ) method
record property

A property on a JSDO table reference that references a JSRecord object with the data for the working record of a temp-table referenced in JSDO local storage. If no working record is set for the referenced temp-table, this property is null.

Data type: JSRecord object
Access: Read-only
Applies to: progress.data.JSDO class, table reference property (JSDO)

The table reference that provides this property can either be the value of a property on the JSDO with the name of a referenced temp-table in JSDO local storage or a reference to the JSDO itself if the JSDO references only a single temp-table.

The field values for the working record are provided by the data property of the JSRecord object returned by the record property.

See also: data property
remove( ) method

Deletes the specified temp-table record referenced in JSDO local storage. The specified record can either be the working record of a referenced temp-table or any record provided by a JSRecord object.

After execution, any working record for an associated temp-table, and for any child temp-tables is not set. To synchronize the change on the AppServer, call the saveChanges( ) method.

Return type: Boolean
Applies to: JSRecord object, progress.data.JSDO class, table reference property (JSDO)

Syntax

```
[table-ref].remove()
jsrecord-ref.remove()
```

table-ref

A table reference on the JSDO for a temp-table that has a working record. If the JSDO references only a single temp-table, the method can be called on the JSDO itself.

jsrecord-ref

A reference to a JSRecord object for a temp-table record in JSDO local storage.

Note: This method does not trigger an automatic sort and has no effect on any existing sort order established for the table reference. However, if there is a sort order that depends on the presence or absence of the record object you are removing, and you want to establish a new sort order with this record object absent, you must manually sort the remaining record objects using the sort( ) method by passing it the same sort function that you used to establish the previous sort order.

The following code fragment shows a jQuery event defined on a delete button to delete the record displayed in a customer detail form from the eCustomer temp-table referenced in JSDO local storage:

```
dataSet = new progress.data.JSDO( 'dsCustomerOrder' );

$('#btnDelete').bind('click', function(event) {
    var jsrecord = dataSet.eCustomer.findById($('#custdetail #id').val());
    jsrecord.remove();
    dataSet.saveChanges();
});
```

The form has been previously displayed with values from the same record. When the button is clicked, the event handler uses the findById( ) method to find the original record with the matching internal record ID (jsrecord) and invokes the remove( ) method on jsrecord to delete the record from eCustomer.

See also: data property, saveChanges( ) method, sort( ) method
response property

Returns an object whose properties contain data from a Mobile built-in or invoke operation executed on the AppServer.

Data type: Object
Access: Read-only
Applies to: request object

If a built-in Mobile operation (create, read, update, or delete) returns successfully and the response is valid JSON that can be converted to a JavaScript object, the response property is a reference to the temp-table or ProDataSet object that is returned from the AppServer. If the server response is not valid JSON, the response property is undefined.

If an invoke operation returns successfully and has no return value or output parameters, the property is null. If the invoke operation has a return value, you can read it as the value of the object _retVal property. If the operation has output parameters, you can read these parameters as the values of object properties whose case-sensitive names and data types match the ABL names and data types of the output parameters specified for the operation on the AppServer.

If the operation returns an ABL error, the object contains the following properties:

- _retVal — A String with the value of any ABL RETURN ERROR string or ReturnValue property for a thrown AppError object
- _errors — An array of JavaScript objects, each of which contains two properties: _errorMsg with the ABL error message string and _errorNum with the error number, for one of possibly many ABL-returned errors

Note: In the current OpenEdge release, this array always returns one object only for the first ABL error (the equivalent of ERROR-STATUS:GET-MESSAGE(1) in ABL).

This request object property is available only for the following events:
- afterCreate
- afterDelete
- afterFill
- afterInvoke
- afterUpdate

See also: fill( ) method, invocation method, saveChanges( ) method
saveChanges( ) method

Synchronizes to the AppServer all changes made to JSDO local storage since the last call to the fill( ) or saveChanges( ) method. The saveChanges( ) method completes this data synchronization by invoking appropriate built-in resource operations on each changed record, one at a time, and in the following general order of operation type:

1. "delete" — All record deletions are applied, one at a time across the network.
2. "create" — The creation of all new records is applied, one at a time across the network.
3. "update" — Updates are applied to all modified records, one at a time across the network.

The sending of changes for multiple operations on the same record is optimized, so the fewest possible changes are sent to the AppServer. For example, if a record is updated, then deleted in local storage, only the deletion is sent to the AppServer.

**Note:** In the current release, there is no batching of built-in record-change operations. That is, the built-in delete operation is invoked over the network for each deleted record, followed by the built-in create operation for each created record, and finally by the built-in update operation for each updated record. So, even for a ProDataSet, a built-in record operation executes over the network only one record at a time and cannot be part of a transaction.

After execution, the working record for each temp-table referenced by the JSDO is not set.

**Return type:** null

**Applies to:** progress.data.JSDO class

**Syntax**

```java
saveChanges( )
```

This method always executes asynchronously, and fires the following JSDO named events, as required by JSDO changes, and shown in operational order:

1. beforeSaveChanges event
2. beforeDelete event
3. afterDelete event
4. beforeCreate event
5. afterCreate event
6. beforeUpdate event
7. afterUpdate event
8. afterSaveChanges event
If any create, update, or delete operation fails, the operation is automatically undone and the applicable record in the JSDO’s local storage reverts to its last-saved state in the database. Specifically:

- If a create operation fails, the record is removed from JSDO memory.
- If an update operation fails, the record reverts to the state it was in immediately preceding the `assign()` operation that led to the failure.
- If a delete operation fails, the record is restored to JSDO memory in its last-saved state. This state does not reflect any unsaved `assign()` operations that may have occurred before the `remove()` call.

You can use the request object returned by the method or event-handler function to retrieve information about the failure and any changes that were discarded as a result:

- To determine where a failure occurred, inspect the `success` and `batch` properties of the request object.
- To retrieve discarded changes, inspect the `jsrecord` property of the request object.

The following code fragment shows a jQuery event defined on a save button to save the current field values for a customer detail form to the corresponding `eCustomer` record in JSDO local storage:

```javascript
dataset = new progress.data.JSDO( 'dsCustomerOrder' );

$('#btnSave').bind('click', function(event) {
  var jsrecord = dataset.eCustomer.findById=$('#custdetail #id').val();
  jsrecord.assign();
  dataset.saveChanges();
});
```

The form has been displayed with previous values of the same record. When the button is clicked, the event handler uses the `findById()` method to find the original record with the matching internal record ID (`jsrecord`) and invokes the `assign()` method on `jsrecord` with an empty parameter list to update its fields in `eCustomer` with any new values entered into the form. It then invokes the `saveChanges()` method to update the corresponding record on the AppServer.

**See also:** `fill()` method, invocation method
services property

Retums an array of objects that identifies the Mobile services that have been loaded for the current Session object and its Mobile Web application.

Data type: Object array
Access: Read-only
Applies to: progress.data.Session class

You load Mobile services for a Session object by loading the corresponding JSDO catalogs using the addCatalog( ) method after you login using the login( ) method.

Each object in the array contains two properties:

- **name** — The name of a Mobile service
- **uri** — The URI for the service. If the address of the service in the catalog is an absolute URI, this value is that URI. If the service address is relative, this value is the relative address concatenated to the value of the Session object’s serviceURI property, which contains the Mobile Web application URI passed to the login( ) method.

Given the following service names and URIs loaded for a Session object:

- "CustomerSvc" service with this URI: 
  
  
  
  "rest/CustomerSvc"

- "ItemSvc" service with this URI:
  
  "http://itemhost:8080/SportsApp/rest/ItemSvc"

The following code fragment produces the output that follows:

```javascript
// create Session
pdsession = new progress.data.Session();

// log in
pdsession.login('/SportsApp');

// load 2 catalogs
pdsession.addCatalog("/SportsApp/static/mobile/CustomerSvc.json");
pdsession.addCatalog("/SportsApp/static/mobile/ItemSvc.json");

// Use services property to print services loaded by this Session object
for (var i=0; i < pdsession.services.length; i++) {
    console.log( pdsession.services[i].name + ' ' +
                pdsession.services[i].uri); }
```

Output from the preceding code fragment:

```
CustomerSvc   /SportsApp/rest/CustomerSvc
ItemSvc       http://itemhost:8080/SportsApp/rest/ItemSvc
```
services property

Note: To return a corresponding list of URIs for the loaded JSDO catalogs, read the catalogURIs property.

See also: addCatalog( ) method, catalogURIs property, login( ) method, serviceURI property
serviceURI property

Returns the URI to the Mobile Web application passed as a parameter to the most recent call to the `login( )` method on the current `Session` object, whether or not the most recent call to `login( )` succeeded.

Data type: String
Access: Read-only
Applies to: `progress.data.Session` class
See also: `loginTarget` property, `login( )` method
setDetailPage() method

Specifies the HTML page that contains the form in which item details are displayed.

Return type: null

Applies to: progress.ui.UIHelper class

Syntax

```
[table-ref.] setDetailPage( object )
```

**table-ref**

A table reference on the UIHelper instance. If the JSDO associated with the UIHelper instance references only a single temp-table, the method can be called on the UIHelper instance itself.

**object**

A JavaScript object with the following properties:

- **name** — The id propety of the HTML div (or other) element that contains the form page
- **fieldTemplate** — (Optional) An HTML string specifying the fields and labels (and their format) included on the form, overriding the default settings or those specified by setFieldTemplate( ).

For example:

```
uiHelper.eCustomer.setDetailPage({
    name: 'cust-detail-page',
    fieldTemplate: '<input id="{__name__}"/>

});
```
setFieldTemplate( ) method

Specifies the format or all detail forms created during the JavaScript session. These settings can be overridden for specific forms by the optional fieldTemplate property of the JavaScript object passed to the setDetailPage( ) method.

Return type: null
Applies to:  progress.ui.UIHelper class

Syntax

```javascript
progress.ui.UIHelper.setFieldTemplate( 'string')
```

string

An HTML string.

It is not necessary to call setFieldTemplate( ) unless:

- You are not using a jQuery Mobile environment, or
- You are using a jQuery Mobile environment, but you want to change the default settings (see below)

OpenEdge Mobile provides a default form template for a jQuery Mobile environment. The following code fragment shows the default template values:

```javascript
progress.ui.UIHelper.setFieldTemplate = '<div data-role="fieldcontain">  <label for="{__name__}">{__label__}</label>  <input id="{__name__}" name="{__name__}" placeholder="" value="" type="text" />
</div>';  
```

As shown in the preceding example, the template uses the following substitution parameters:

- `{__name__}` — The name of the field, as defined by the schema
- `{__label__}` — The title property of the field, as defined by the schema

See also:  setDetailPage( ) method
setItemTemplate( ) method

Specifies the format for items in all list views created during the JavaScript session. These settings can be overridden for specific list views by setListView( ). In addition, the __format__ parameter can be overridden for specific items by addItem( ).

**Return type:** null

**Applies to:** progress.ui.UIHelper class

**Syntax**

```
progress.ui.UIHelper.setItemTemplate( 'string')
```

```
string

An HTML string.
```

It is not necessary to call setItemTemplate( ) unless:

- You are not using a jQuery Mobile environment, or
- You are using a jQuery Mobile environment, but you want to change the default settings (see below)

OpenEdge Mobile provides a default item template for a jQuery Mobile environment. The following code fragment shows the default template values:

```
progress.ui.UIHelper.setItemTemplate = 'li data-theme="c"
data-id="{__id__}"<a href="#{__page__}" class="ui-link"data-transition="slide">{__format__}</a></li>''
```

As shown in the preceding example, the template uses the following substitution parameters:

- __id__ — The internal ID of the record
- __page__ — The name attribute of the object passed as a parameter to setDetailPage( ), which defines the form for each individual list item
- __format__ — The format property of the object passed as a parameter to setListView( ) or (optionally) to addItem( ), which identifies the fields to be included for each item in the list view

**See also:** setListView( ) method, addItem( ) method
setListView( ) method

Defines a list view for a given table.

**Return type:** null

**Applies to:** progress.ui.UIHelper class, table reference property (UIHelper)

**Syntax**

```
[table-ref.]setListView( object )
```

**table-ref**

A table reference on the UIHelper instance. If the JSDO associated with the UIHelper instance references only a single temp-table, the method can be called on the UIHelper instance itself.

**object**

A JavaScript object that contains one or more of the following properties:

- **name** — A string that identifies the list view.
- **format** — (Optional) A string that specifies the fields shown for each item and the line breaks between them. If the format property is omitted, the UIHelper instance uses the list item in the HTML ListView as a template.
- **autoLink** — A Boolean that specifies whether events are automatically generated to display the corresponding form on the detail page when an item is clicked.
- **itemTemplate** — An HTML string that overrides, for this list view, the settings in the default item template or in the 'string' parameter passed to the setItemTemplate( ) method.

For example:

```
uihelper.eCustomer.setListView({
    name: 'listview',
    format: '{CustNum}<br>{Name}<br>{Phone}<br>{Address}<br>{State}',
    autoLink: true
});
```

**See also:** setItemTemplate( ) method
setSortFields( ) method

Specifies or clears the record fields on which to automatically sort the record objects for a table reference after you have set its autoSort property to true (the default). This method enables or disables automatic sorting based on record fields only for supported JSDO operations. See the description of the autoSort property for more information.

After completing execution, this method has no effect on the working record for the affected table reference.

Return type: null
Applies to: progress.data.JSDO class, table reference property (JSDO)

Syntax

```
[table-ref.]setSortFields ( sort-fields )
```

table-ref

A table reference on the JSDO. If the JSDO references only a single temp-table, the method can be called on the JSDO itself.

sort-fields

An array of String values set to the names of record fields on which to sort the record objects, with an optional indication of the sort order for each field. This array can have the following syntax:

Syntax

```
[ "field-name[;sort-order]", "field-name[;sort-order]", ... ]
```

field-name

The name of a field in the record objects of the specified table reference. A field-name can have the OpenEdge-reserved name, "_id". Otherwise, any field-name must already exist in the JSDO schema and must have a scalar value (cannot be an array field).

sort-order

An indication of the sort order for the field, which can have one of the following case-insensitive values:

- **ASC** — Sorts ascending.
- **ASCENDING** — Sorts ascending.
- **DESC** — Sorts descending.
- **DESCENDING** — Sorts descending.

The default sort order is ascending.
When the automatic sort occurs, the record objects are sorted and grouped by each successive field-name in the array, according to its JavaScript data type and specified sort-order. Fields are compared using the $\gt$, $\lt$, and $\neq$ JavaScript operators. String fields can be compared with or without case sensitivity depending on the caseSensitive property setting. However, note that date fields are compared as dates, even though they are represented as strings in JavaScript.

If you set the sort-fields parameter to null, or you specify an empty array, the method clears all sort fields. Automatic sorting for table-ref can then occur only if there is an existing sort function setting using the setSortFn( ) method.

**Note:** If you set a sort function for the table reference using setSortFn( ) in addition to using this method to set sort fields, the sort function takes precedence.

In the following code fragment, assuming the autoSort property is set to true on dsCustomer.eCustomer (the default), after the fill( ) method initializes JSDO local storage, the record objects for eCustomer are sorted by the Country field ascending, then by the State field within Country ascending, then by the Balance field within State descending. At a later point, the foreach( ) method then loops through these record objects, starting with the first record in eCustomer sort order:

```javascript
dsCustomer = new progress.data.JSDO( { name: 'dsCustomer' });
dsCustomer.fill();
... 
dsCustomer.eCustomer.foreach( function( customer ){ ... } );
```

See also: autoSort property, caseSensitive property, setSortFn( ) method, sort( ) method
setSortFn( ) method

Specifies or clears a user-defined sort function with which to automatically sort the record objects for a table reference after you have set its `autoSort` property to `true` (the default). This method enables or disables automatic sorting based on a sort function only for supported JSDO operations. See the description of the `autoSort` property for more information.

After completing execution, this method has no effect on the working record for the affected table reference.

Return type: null
Applies to: progress.data.JSDO class, table reference property (JSDO)

Syntax

```javascript
[table-ref.]setSortFn ( funcRef )
```

table-ref

A table reference on the JSDO. If the JSDO references only a single temp-table, the method can be called on the JSDO itself.

funcRef

A reference to a JavaScript sort function that compares two record objects for the sort and returns a Number value. This function must have following signature:

Syntax

```javascript
function [ func-name ] ( jsrecord-ref1 , jsrecord-ref2 )
```

Where `func-name` is the name of a function that you define external to the `setSortFn( )` method parameter list, and `jsrecord-ref1` and `jsrecord-ref2` are two JSRecord objects that the function compares from the specified table reference. You can then pass `func-name` to the `setSortFn( )` method as the `funcRef` parameter. Alternatively, you can specify `funcRef` as the entire inline function definition without `func-name`.

Your `funcRef` code determines the criteria by which one of the two input record objects follows the other in the sort order, and returns one of the following values depending on the result:

- `1` — The `jsrecord-ref1` object follows (is “greater than”) the `jsrecord-ref2` object in the sort order.
- `-1` — The `jsrecord-ref1` object precedes (is “less than”) the `jsrecord-ref2` object in the sort order.
- `0` — The two record objects occupy the same position (are “equal”) in the sort order.
When OpenEdge calls an automatic sort, and a sort function is set using this method, the sort uses this function to determine the sort order for every pair of records that it tests as it iterates through the record objects of the specified table reference.

If you set the `funcRef` parameter to `null`, the method clears any sort function definition. Automatic sorting for the table reference can then occur only if there are one or more existing sort fields set using the `setSortFields()` method.

**Notes:** Any default JavaScript comparisons that you make with `String` fields in `funcRef` are case sensitive according to JavaScript rules and ignore the setting of the `caseSensitive` property.

If you set sort fields for the table reference using `setSortFields()` in addition to using this method to set a sort function, the sort function takes precedence.

In the following code fragment, assuming the `autoSort` property is set to `true` on `dsCustomer.eCustomer` (the default), after the `fill()` method initializes JSDO local storage, the record objects for `eCustomer` are automatically sorted using the results of the external user-defined function, `sortOnNameCSensitive()`, whose reference is passed to the `setSortFn()` method. In this case, the function compares the case-sensitive values of the `Name` fields from each pair of `eCustomer` record objects selected by the sort. At a later point, the `foreach()` method then loops through these record objects, starting with the first record in `eCustomer` sort order:

```javascript
dsCustomer = new progress.data.JSDO( { name: 'dsCustomer' });
dsCustomer.setSortFn( sortOnNameCSensitive );
dsCustomer.fill();
...;
dsCustomer.eCustomer.foreach( function( customer ){ . . . } );

function sortOnNameCSensitive ( rec1 , rec2 ) {
    if (rec1.data.Name > rec2.data.Name)
        return 1;
    else if (rec1.data.Name < rec2.data.Name)
        return -1;
    else
        return 0;
}
```

If you want to compare the `Name` field in this function using a case-insensitive test, you can use the JavaScript `toUpperCase()` function in the user-defined function. For example, in `sortOnNameCInsensitive()`, as follows:

```javascript
dsCustomer = new progress.data.JSDO( { name: 'dsCustomer' });
dsCustomer.setSortFn( sortOnNameCInsensitive );
dsCustomer.fill();
...;
dsCustomer.eCustomer.foreach( function( customer ){ . . . } );

function sortOnNameCInsensitive ( rec1 , rec2 ) {
    if (rec1.data.Name.toUpperCase() > rec2.data.Name.toUpperCase())
        return 1;
    else if (rec1.data.Name.toUpperCase() < rec2.data.Name.toUpperCase())
        return -1;
    else
        return 0;
```
See also:

autoSort property, caseSensitive property, setSortFields() method, sort() method
**showListView() method**

Displays the specified table’s list view on the Mobile App device or browser.

**Return type:** null

**Applies to:** progress.ui.UIHelper class, table reference property (UIHelper)

**Syntax**

```
[table-ref.] showListView()
```

**table-ref**

A table reference on the UIHelper instance. If the JSDO associated with the UIHelper instance references only a single temp-table, the method can be called on the UIHelper instance itself.

**See also:** setListView() method
sort( ) method

Sorts the existing record objects for a table reference in JSDO local storage using either specified sort fields or a specified user-defined sort function.

After completing execution, this method has no effect on the working record for the affected table reference.

Return type: null
Applies to: progress.data.JSDO class, table reference property (JSDO)

Syntax

```
[table-ref].sort ( { sort-fields | funcRef } )
```

**table-ref**

A table reference on the JSDO. If the JSDO references only a single temp-table, the method can be called on the JSDO itself.

**sort-fields**

An array of String values set to the names of record fields on which to sort the record objects, with an optional indication of the sort order for each field. This array can have the following syntax:

**Syntax**

```
[ "field-name[;sort-order]" [, "field-name[;sort-order]" ] ... ]
```

**field-name**

The name of a field in the record objects of the specified table reference. A field-name can have the OpenEdge-reserved name, "_id". Otherwise, any field-name must already exist in the JSDO schema and must have a scalar value (cannot be an array field).

**sort-order**

An indication of the sort order for the field, which can have one of the following case-insensitive values:

- **ASC** — Sorts ascending.
- **ASCENDING** — Sorts ascending.
- **DESC** — Sorts descending.
- **DESCENDING** — Sorts descending.

The default sort order is ascending.

When the sort occurs, the record objects are sorted and grouped by each successive field-name in the array, according to its JavaScript data type and specified sort-order. Fields are compared using the >, <, and = JavaScript
operators. String fields can be compared with or without case sensitivity depending on the caseSensitive property setting. However, note that date fields are compared as dates, even though they are represented as strings in JavaScript.

**funcRef**

A reference to a JavaScript sort function that compares two record objects for the sort and returns a Number value. This function must have following signature:

**Syntax**

```javascript
function [ func-name ] ( jsrecord-ref1 , jsrecord-ref2 )
```

Where `func-name` is the name of a function that you define external to the `sort()` method parameter list, and `jsrecord-ref1` and `jsrecord-ref2` are two JSRecord objects that the function compares from the specified table reference. You can then pass `func-name` to the `sort()` method as the `funcRef` parameter. Alternatively, you can specify `funcRef` as the entire inline function definition without `func-name`.

Your `funcRef` code determines the criteria by which one of the two input record objects follows the other in the sort order, and returns one of the following values depending on the result:

- **1** — The `jsrecord-ref1` object follows (is “greater than”) the `jsrecord-ref2` object in the sort order.
- **-1** — The `jsrecord-ref1` object precedes (is “less than”) the `jsrecord-ref2` object in the sort order.
- **0** — The two record objects occupy the same position (are “equal”) in the sort order.

When you invoke the `sort()` method with a sort function, the sort uses this function to determine the sort order for every pair of records that it tests as it iterates through the record objects of the specified table reference.

**Note:** Any default JavaScript comparisons that you make with String fields in `funcRef` are case sensitive according to JavaScript rules and ignore the setting of the caseSensitive property.

In the following code fragment, the `fill()` method initializes JSDO local storage with `eCustomer` record objects from the AppServer in order of the temp-table primary key (the default). The `sort()` method later sorts the record objects for `eCustomer` by the Country field ascending, then by the State field within Country ascending, then by the Balance field within State descending. The `foreach()` function then loops through these record objects in the new `eCustomer` sort order:
In the following code fragment, the `fill()` method initializes JSDO local storage with eCustomer record objects from the AppServer in order of the temp-table primary key (the default). The `sort()` method later sorts the record objects for eCustomer using the results of an inline function definition, which in this case compares the case-sensitive values of the Name fields from each pair of eCustomer record objects selected by the sort. The `foreach()` method then loops through these record objects in the new eCustomer sort order:

```javascript
dsCustomer = new progress.data.JSDO( { name: 'dsCustomer' });
dsCustomer.fill();
...
dsCustomer.sort( [ "Country", "State", "Balance:DESC" ] );
dsCustomer.eCustomer.foreach( function( customer ){ ... } );
```

If you want to compare the Name fields using a case-insensitive test, you can use the JavaScript `toUpperCase()` function in the inline function definition, as follows:

```javascript
dsCustomer = new progress.data.JSDO( { name: 'dsCustomer' });
dsCustomer.fill();
...
dsCustomer.sort( function ( rec1 , rec2 ) {
    if (rec1.data.Name > rec2.data.Name)
        return 1;
    else if (rec1.data.Name < rec2.data.Name)
        return -1;
    else
        return 0;
}
);
dsCustomer.eCustomer.foreach( function( customer ){ ... } );
```

Note: Because the `sort()` method executes in JavaScript on the client side, sorting a large set of record objects can take a significant amount of time and make the UI appear to be locked. You might set a wait or progress indicator just prior to invoking the sort to alert the user that the app is working.

See also: autoSort property, caseSensitive property, setSortFields() method, setSortFn() method
subscribe( ) method

Subscribes a given event handler function to a named event for a JSDO or table reference. For more information on these events, see Table 13.

After execution, the working record for any associated table reference remains unchanged.

**Return type:** null

**Applies to:** progress.data.JSDO class, table reference property (JSDO)

**Syntax**

```
[table-ref].subscribe( event-name [ , op-name ] , event-handler , scope )
```

**table-ref**

A table reference on the JSDO, which allows the method to subscribe an event handler for events that fire only for a referenced temp-table. If invoked on the JSDO directly, the method can subscribe an event handler for all JSDO events.

**event-name**

The name of an event to which the JSDO instance or table-ref subscribes. See Table 13 for a list of available JSDO events.

If you call the subscribe( ) method on table-ref, the method can subscribe only to the following events:

- afterCreate
- afterDelete
- afterUpdate
- beforeCreate
- beforeDelete
- beforeUpdate

**op-name**

The name of a JSDO invocation method, a call to which causes the event to fire. This parameter is required in cases where event-name is beforeInvoke or afterInvoke. It should be used only with those event names, and only when subscribing the JSDO (that is, not a table-ref). The value of op-name is the same as that of the fnName property of the request object.

**event-handler**

A reference to an event handler function that is called when the event fires.
**scope**

An optional object reference that defines the execution scope of the function called when the event fires. If the `scope` property is omitted, the execution scope is the global object (usually the browser or device window).

See also: invocation method, unsubscribe( ) method, Table 13
success property

A Boolean that indicates, if set to true, that the Mobile operation was successfully executed. A successful Mobile operation returns an HTTP status in the range of 200 - 299.

**Data type:** Boolean  
**Access:** Read-only  
**Applies to:** request object

This request object property is available only for the following events:

- afterCreate
- afterDelete
- afterFill
- afterInvoke
- afterSaveChanges
- afterUpdate

In the case of an afterSaveChanges event, this property is true only if all requests that are part of the batch of requests were successfully executed.

**See also:** fill( ) method, invocation method, saveChanges( ) method
An object reference property on a JSDO that has the name of a temp-table mapped by the Mobile resource for which the current JSDO is created. Its value is a reference (table reference) to the temp-table object in JSDO local storage. This temp-table object provides access to the working record, if defined, of the temp-table. If a Mobile resource maps a ProDataSet, its JSDO provides one table reference for every temp-table in the ProDataSet.

**Data type:** Temp-table object reference in JSDO local storage

**Access:** Read-only

**Applies to:** progress.data.JSDO class

In syntax wherever a table reference can be used, `table-ref` represents the name of the property containing the table reference. A referenced temp-table object provides the following properties:

- **record property** — A reference to a JSRecord object that has the data for the working record of the temp-table through a `data` property. If no working record is defined for the temp-table, the `record` property is `null`.

- **field reference property** — A property on the temp-table object that has the name of a field (as defined in the temp-table schema) and the value for that field in the working record, also referred to as a field reference. In syntax wherever a field reference can be used, `field-ref` represents the name of the property containing the field reference. A temp-table object provides one field reference for each field defined in the original temp-table. If no working record is defined, all field references are `null`, except when fields are referenced on the `data` property of a JSRecord object reference.

**Caution:** Never write directly to a `field-ref` using this `record` property or any JSRecord object reference; in this case, use `field-ref` only to read the data. Writing to data using such a reference does not mark the record for update when calling the `saveChanges()` method, nor does it re-sort the record in local storage according to any order you have established using the `autoSort` property. To mark a record for update and automatically re-sort the record according to the `autoSort` property, you must assign a field value either by setting a `jsdo.table-ref.field-ref` for a working record or by calling the `assign()` method on a valid `table-ref` or JSRecord object reference.

You can therefore reference the field values in the working record of a given table reference using either the JSRecord object returned by its `record` property or its field references. The field references provide a convenient way to access table fields that is similar to how you reference table fields in ABL. The `record` property provides an alternative way to access table fields, especially one that has the same name as a property or method of a JSDO that you can invoke from a table reference.

For example, the following code fragment shows two different ways to access the `CustNum` field of a record added to a customer temp-table provided by a resource ProDataSet:
Both calls to the `alert()` function access the same `CustNum` field in the working record of the `customer` table created by the `add()` method.

For more information on accessing the working record of a table reference, see the notes for the section on the `progress.data.JSDO` class.

**See also:**  
`autoSort` property, `JSRecord` object, `record` property

```javascript
var dataSet = new Progress.data.JSDO( 'CustomerOrderDS' );
dataset.customer.add();
alert(dataSet.customer.record.data.CustNum);
alert(dataSet.customer.CustNum);
```
table reference property (UIHelper)

An object reference property on a UIHelper instance that corresponds to a temp-table of the JSDO with which the instance is associated.

**Data type:** Temp-table object reference in JSDO local storage  
**Access:** Read-only  
**Applies to:** progress.ui.UIHelper class

**Note:** A table reference (table-ref in syntax) on a UIHelper object does not have a record property. Therefore you cannot use this table reference to directly access temp-table field values, but only to invoke methods on the UIHelper object.

**See also:** table reference property (JSDO)
unsubscribe( ) method

Unsubscribes a given event handler function from a named event of a JSDO or table reference. For more information on these events, see Table 13.

After execution, the working record for any associated table reference remains unchanged.

Return type: null

Applies to: progress.data.JSDO class, table reference property (JSDO)

Syntax

```
[table-ref.] unsubscribe ( event-name [ , op-name ] , event-handler , scope )
```

**table-ref**

A table reference on the JSDO, which allows the method to unsubscribe an event handler for events that fire only for a referenced temp-table. If invoked on the JSDO, the method can unsubscribe an event handler for all JSDO events.

**event-name**

If you call the unsubscribe( ) method on table-ref, the method can unsubscribe only from the following events:

- afterCreate
- afterDelete
- afterUpdate
- beforeCreate
- beforeDelete
- beforeUpdate

**op-name**

The name of a JSDO invocation method, a call to which causes the event to fire. This parameter is required in cases where event-name is beforeInvoke or afterInvoke, and op-name should be used only with those event names. To be meaningful, the op-name value must match that of the corresponding parameter in a preceding subscribe( ) call.

**event-handler**

A reference to an event handler function that is called when the event fires.

**scope**

An object reference that defines the execution scope of the specified event-handler. Specifying the scope is optional in the event subscription. If the subscription does specify an execution scope, you must specify a matching scope parameter to the unsubscribe( ) method.
See also: invocation method, subscribe( ) method, Table 13
useRelationships property

A Boolean that specifies whether JSDO methods that operate on table references in JSDO local storage work with the table relationships defined in the schema (that is, work only on the records of a child table that are related to the parent).

**Data type:** Boolean

**Access:** Writable

**Applies to:** progress.data.JSDO class

When set to true, methods, such as add( ), find( ), and foreach( ), that have default behavior for related table references respect these relationships when operating on related tables. When set to false, these methods operate on all table references as if they have no relationships. The default value is true.

**See also:** add( ) method, find( ) method, foreach( ) method
**userName property**

Returns the user ID passed as a parameter to the most recent call to the `login()` method on the current `Session` object.

**Data type:** String  
**Access:** Read-only  
**Applies to:** `progress.data.Session` class

This value is returned, whether or not the most recent call to `login()` succeeded.

**Note:** This property does not always specify the name of the user logged in for the current session. The logged-in user can be different from this property setting if the authentication was done by the browser or hybrid native wrapper prior to the `login()` method being called, or if the `login()` method was passed incorrect user credentials and the browser or native wrapper took over and completed the user authentication.

**See also:** `login()` method
xhr property

A reference to the XMLHttpRequest object used to perform a Mobile operation request. In the case of an asynchronous call, this property may not be available until after the XMLHttpRequest object is created.

**Data type:** Object  
**Access:** Read-only  
**Applies to:** request object

This request object property is available only for the following events:

- afterCreate
- afterDelete
- afterFill
- afterInvoke
- afterUpdate

**See also:** fill( ) method, invocation method, saveChanges( ) method
Third party acknowledgements

One or more products in the Progress OpenEdge v11.3 release includes third party components covered by licenses that require that the following documentation notices be provided:

Progress OpenEdge v11.3 may incorporate ANT v1.5.4. Such technology is subject to the following terms and conditions: The Apache Software License, Version 1.1, applies to all versions of up to ant 1.6.0 included. The Apache Software License, Version 1.1 - Copyright (C) 2000-2003 The Apache Software Foundation. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: 1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. 2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. 3. The end-user documentation included with the redistribution, if any, must include the following acknowledgment: "This product includes software developed by the Apache Software Foundation (http://www.apache.org/)." Alternately, this acknowledgment may appear in the software itself, if and wherever such third-party acknowledgments normally appear. 4. The names "Ant" and "Apache Software Foundation" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact apache@apache.org. 5. Products derived from this software may not be called "Apache", nor may "Apache" appear in their name, without prior written permission of the Apache Software Foundation. THIS SOFTWARE IS PROVIDED "AS IS" AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE APACHE SOFTWARE FOUNDATION OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN
ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. This software consists of voluntary contributions made by many individuals on behalf of the Apache Software Foundation. For more information on the Apache Software Foundation, please see <http://www.apache.org/>.

Progress OpenEdge v11.3 may incorporate Apache Avalon 2002/08/06, Apache Batik v1.1, Apache Jakarta POI v1.1, Apache POI 2003/05/14, Regexp (Pure Java Regular Expression) v1.2, and XMLTask v1.15.1 from The Apache Software Foundation. Such technologies are subject to the following terms and conditions: The Apache Software License, Version 1.1 Copyright (c) 2000 The Apache Software Foundation. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: 1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. 2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. 3. The end-user documentation included with the redistribution, if any, must include the following acknowledgment: "This product includes software developed by the Apache Software Foundation (http://www.apache.org/)." Alternately, this acknowledgment may appear in the software itself, if and wherever such third-party acknowledgments normally appear. 4. The names "Apache" and "Apache Software Foundation" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact apache@apache.org. 5. Products derived from this software may not be called "Apache", nor may "Apache" appear in their name, without prior written permission of the Apache Software Foundation. THIS SOFTWARE IS PROVIDED "AS IS" AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE APACHE SOFTWARE FOUNDATION OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. This software consists of voluntary contributions made by many individuals on behalf of the Apache Software Foundation. For more information on the Apache Software Foundation, please see <http://www.apache.org/>.

Progress OpenEdge v11.3 may incorporate Apache JAXP v1.3.04 from The Apache Software Foundation. Such technology is subject to the following terms and conditions: The Apache Software License, Version 1.1 Copyright (c) 1999-2003 The Apache Software Foundation. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: 1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. 2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. 3. The end-user documentation included with the redistribution, if any, must include the following acknowledgment: "This product includes software developed by the Apache Software Foundation (http://www.apache.org/)." Alternately, this acknowledgment may appear in the software itself, if and wherever such third-party acknowledgments
normally appear. 4. The names "Xalan" and "Apache Software Foundation" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact apache@apache.org. 5. Products derived from this software may not be called "Apache", nor may "Apache" appear in their name, without prior written permission of the Apache Software Foundation. THIS SOFTWARE IS PROVIDED ``AS IS'' AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE APACHE SOFTWARE FOUNDATION OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. This software consists of voluntary contributions made by many individuals on behalf of the Apache Software Foundation and was originally based on software copyright (c) 1999, Lotus Development Corporation., http://www.lotus.com. For more information on the Apache Software Foundation, please see http://www.apache.org/.

Progress OpenEdge v11.3 may incorporate Crimson v1.1.3 from Progress Extensions for Eclipse v2.2.1. Such technology is subject to the following terms and conditions: The Apache Software License, Version 1.1 Copyright (c) 1999-2003 The Apache Software Foundation. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: 1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. 2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. 3. The end-user documentation included with the redistribution, if any, must include the following acknowledgment: "This product includes software developed by the Apache Software Foundation (http://www.apache.org/)." Alternately, this acknowledgment may appear in the software itself, if and wherever such third-party acknowledgments normally appear. 4. The names "Xerces" and "Apache Software Foundation" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact apache@apache.org. 5. Products derived from this software may not be called "Apache", nor may "Apache" appear in their name, without prior written permission of the Apache Software Foundation. THIS SOFTWARE IS PROVIDED ``AS IS'' AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE APACHE SOFTWARE FOUNDATION OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. This software consists of voluntary contributions made by many individuals on behalf of the Apache Software Foundation and was originally based on software copyright (c) 1999, International Business Machines, Inc., http://www.ibm.com. For more information on the Apache Software Foundation, please see <http://www.apache.org/>. 
Progress OpenEdge v11.3 may incorporate jfor v0.7.2 from jfor. Such technology is subject to the following terms and conditions: jfor Apache-Style Software License. Copyright (c) 2002 by the jfor project. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: 1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. 2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. 3. The end-user documentation included with the redistribution, if any, must include the following acknowledgment: "This product includes software developed by the jfor project (http://www.jfor.org)." Alternately, this acknowledgment may appear in the software itself, if and wherever such third-party acknowledgments normally appear. 4. The name "jfor" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact info@jfor.org. 5. Products derived from this software may not be called "jfor", nor may "jfor" appear in their name, without prior written permission of info@jfor.org. THIS SOFTWARE IS PROVIDED "AS IS" AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE JFOR PROJECT OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Progress OpenEdge v11.3 may incorporate Apache LogKit v1.2 from The Apache Software Foundation. Such technologies are subject to the following terms and conditions: The Apache Software License, Version 1.1 -- Copyright (C) 1997-2003 The Apache Software Foundation. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: 1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. 2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. 3. The end-user documentation included with the redistribution, if any, must include the following acknowledgment: "This product includes software developed by the Apache Software Foundation (http://www.apache.org/)." Alternately, this acknowledgment may appear in the software itself, if and wherever such third-party acknowledgments normally appear. 4. The names "LogKit", "Jakarta" and "Apache Software Foundation" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact apache@apache.org. 5. Products derived from this software may not be called "Apache", nor may "Apache" appear in their name, without prior written permission of the Apache Software Foundation. THIS SOFTWARE IS PROVIDED "AS IS" AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE APACHE SOFTWARE FOUNDATION OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. This software consists of voluntary contributions made by many individuals on behalf of the Apache Software Foundation. For more information on the Apache Software Foundation, please see <http://www.apache.org/>.

Progress OpenEdge v11.3 may incorporate Xerces for Java XML Parser v2.6.2. Such technology is subject to the following terms and conditions: The Apache Software License, Version 1.1 Copyright (c) 1999 The Apache Software Foundation. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: 1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. 2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. 3. The end-user documentation included with the redistribution, if any, must include the following acknowledgment: "This product includes software developed by the Apache Software Foundation (http://www.apache.org/)." Alternately, this acknowledgment may appear in the software itself, if and wherever such third-party acknowledgments normally appear. 4. The names "Xerces" and "Apache Software Foundation" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact apache@apache.org. 5. Products derived from this software may not be called "Apache", nor may "Apache" appear in their name, without prior written permission of the Apache Software Foundation. THIS SOFTWARE IS PROVIDED \"AS IS\" AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE APACHE SOFTWARE FOUNDATION OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. This software consists of voluntary contributions made by many individuals on behalf of the Apache Software Foundation and was originally based on software copyright (c) 1999, International Business Machines, Inc., http://www.ibm.com. For more information on the Apache Software Foundation, please see <http://www.apache.org/>.

Progress OpenEdge v11.3 may incorporate SOAP v2.3.1 from Apache Foundation. Such technology is subject to the following terms and conditions: The Apache Software License, Version 1.1

Copyright (c) 1999 The Apache Software Foundation. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
Appendix D: Third party acknowledgements

3. The end-user documentation included with the redistribution, if any, must include the following acknowledgment: "This product includes software developed by the Apache Software Foundation (http://www.apache.org)."

Alternately, this acknowledgment may appear in the software itself, if and wherever such third-party acknowledgments normally appear.

4. The names "SOAP" and "Apache Software Foundation" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact apache@apache.org.

5. Products derived from this software may not be called "Apache", nor may "Apache" appear in their name, without prior written permission of the Apache Software Foundation.

THIS SOFTWARE IS PROVIDED "AS IS" AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE APACHE SOFTWARE FOUNDATION OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

This software consists of voluntary contributions made by many individuals on behalf of the Apache Software Foundation. For more information on the Apache Software Foundation, please see <http://www.apache.org>.

Progress OpenEdge v11.3 may incorporate ANTLR (Another Tool for Language Recognition) v2.7.6. Such technology is subject to the following terms and conditions: ANTLR 3 License [The BSD License] Copyright (c) 2003-2006, Terence Parr All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. Neither the name of the author nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Progress OpenEdge v11.3 may incorporate Castor v0.9.3 from castor.org. Such technology is subject to the following terms and conditions: Original Intalio license -
Copyright 1999-2004 (C) Intalio Inc., and others. All Rights Reserved. Redistribution and use of this software and associated documentation ("Software"), with or without modification, are permitted provided that the following conditions are met: 1. Redistributions of source code must retain copyright statements and notices. Redistributions must also contain a copy of this document. 2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. 3. The name "ExoLab" must not be used to endorse or promote products derived from this Software without prior written permission of Intalio Inc. For written permission, please contact info@exolab.org. 4. Products derived from this Software may not be called "Castor" nor may "Castor" appear in their names without prior written permission of Intalio Inc. Exolab, Castor and Intalio are trademarks of Intalio Inc. 5. Due credit should be given to the ExoLab Project (http://www.exolab.org/). THIS SOFTWARE IS PROVIDED BY INTALIO AND CONTRIBUTORS "AS IS" AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL INTALIO OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Progress OpenEdge v11.3 may incorporate Decimal Conversion Code (dtoa.c; g_fmt.c; md_prod.s; decstrd.c; decstrtof.c; dmisc.c; gdtrea.c; gdtrea.h; gdtreamp.h; gethex.c; gmisc.c; hd_init.c; misc.c; smisc.c; strtodg.c; strtod.c; sum.c; ulp.c). Such technologies are subject to the following terms and conditions: dtoa.c License: The author of this software is David M. Gay. Copyright (c) 1991, 2000, 2001 by Lucent Technologies. Permission to use, copy, modify, and distribute this software for any purpose without fee is hereby granted, provided that this entire notice is included in all copies of any software which is or includes a copy or modification of this software and in all copies of the supporting documentation for such software. THIS SOFTWARE IS BEING PROVIDED "AS IS", WITHOUT ANY EXPRESS OR IMPLIED WARRANTY. IN PARTICULAR, NEITHER THE AUTHOR NOR LUCENT MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND CONCERNING THE MERCHANTABILITY OF THIS SOFTWARE OR ITS FITNESS FOR ANY PARTICULAR PURPOSE. g_fmt.c License: The author of this software is David M. Gay. Copyright (c) 1991, 1996 by Lucent Technologies. Permission to use, copy, modify, and distribute this software for any purpose without fee is hereby granted, provided that this entire notice is included in all copies of any software which is or includes a copy or modification of this software and in all copies of the supporting documentation for such software. THIS SOFTWARE IS BEING PROVIDED "AS IS", WITHOUT ANY EXPRESS OR IMPLIED WARRANTY. IN PARTICULAR, NEITHER THE AUTHOR NOR LUCENT MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND CONCERNING THE MERCHANTABILITY OF THIS SOFTWARE OR ITS FITNESS FOR ANY PARTICULAR PURPOSE.

md_prod.s License: The author of this software is David M. Gay. Copyright (c) 1991 by Lucent Technologies. Permission to use, copy, modify, and distribute this software for any purpose without fee is hereby granted, provided that this entire notice is included in all copies of any software which is or includes a copy or modification of this software and in all copies of the supporting documentation for such software. THIS SOFTWARE IS BEING PROVIDED "AS IS", WITHOUT ANY EXPRESS OR IMPLIED WARRANTY. IN PARTICULAR, NEITHER THE AUTHOR NOR LUCENT MAKES ANY
Appendix D: Third party acknowledgements

REPRESENTATION OR WARRANTY OF ANY KIND CONCERNING THE
MERCHANTABILITY OF THIS SOFTWARE OR ITS FITNESS FOR ANY
PARTICULAR PURPOSE. decstrtod.c License: The author of this software is David M.
Gay. Copyright (C) 1998-2001 by Lucent Technologies All Rights Reserved Permission
to use, copy, modify, and distribute this software and its documentation for any purpose
and without fee is hereby granted, provided that the above copyright notice appear in
all copies and that both that the copyright notice and this permission notice and
warranty disclaimer appear in supporting documentation, and that the name of Lucent
or any of its entities not be used in advertising or publicity pertaining to distribution of
the software without specific, written prior permission. LUCENT DISCLAIMS ALL
WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED
WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL
LUCENT OR ANY OF ITS ENTITIES BE LIABLE FOR ANY SPECIAL, INDIRECT OR
CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING
FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF
CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR
IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

decstrtof.c License: The author of this software is David M. Gay. Copyright (C) 1998,
2000 by Lucent Technologies All Rights Reserved Permission to use, copy, modify,
and distribute this software and its documentation for any purpose and without fee is
hereby granted, provided that the above copyright notice appear in all copies and that
both that the copyright notice and this permission notice appear in supporting
documentation, and that the name of Lucent or any of its entities not be used in advertising or publicity
pertaining to distribution of the software without specific, written prior permission.
LUCENT DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL LUCENT OR ANY OF ITS ENTITIES BE LIABLE FOR ANY SPECIAL, INDIRECT OR
CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF
CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
dmisc.c License: The author of this software is David M. Gay. Copyright (C) 1998 by Lucent Technologies All Rights
Reserved Permission to use, copy, modify, and distribute this software and its
documentation for any purpose and without fee is hereby granted, provided that the
above copyright notice appear in all copies and that both that the copyright notice and
this permission notice and warranty disclaimer appear in supporting documentation,
and that the name of Lucent or any of its entities not be used in advertising or publicity
pertaining to distribution of the software without specific, written prior permission.
LUCENT DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE,
INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS.
IN NO EVENT SHALL LUCENT OR ANY OF ITS ENTITIES BE LIABLE FOR ANY
SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS,
WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS
ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR
PERFORMANCE OF THIS SOFTWARE.
gdtoa.c License: The author of this software is David M. Gay. Copyright (C) 1998, 1999 by Lucent Technologies All Rights
Reserved Permission to use, copy, modify, and distribute this software and its
documentation for any purpose and without fee is hereby granted, provided that the
above copyright notice appear in all copies and that both that the copyright notice and
this permission notice and warranty disclaimer appear in supporting documentation,
and that the name of Lucent or any of its entities not be used in advertising or publicity
pertaining to distribution of the software without specific, written prior permission.
LUCENT DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE,
INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS.
IN NO EVENT SHALL LUCENT OR ANY OF ITS ENTITIES BE LIABLE FOR ANY
SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

gdtoa.h License: The author of this software is David M. Gay. Copyright (C) 1998 by Lucent Technologies All Rights Reserved Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that the copyright notice and this permission notice and warranty disclaimer appear in supporting documentation, and that the name of Lucent or any of its entities not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission. LUCENT DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL LUCENT OR ANY OF ITS ENTITIES BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

gdtoaimp.h License: The author of this software is David M. Gay. Copyright (C) 1998-2000 by Lucent Technologies All Rights Reserved Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that the copyright notice and this permission notice and warranty disclaimer appear in supporting documentation, and that the name of Lucent or any of its entities not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission. LUCENT DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL LUCENT OR ANY OF ITS ENTITIES BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

gethex.c License: The author of this software is David M. Gay. Copyright (C) 1998 by Lucent Technologies All Rights Reserved Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that the copyright notice and this permission notice and warranty disclaimer appear in supporting documentation, and that the name of Lucent or any of its entities not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission. LUCENT DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL LUCENT OR ANY OF ITS ENTITIES BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

gmisc.c License: The author of this software is David M. Gay. Copyright (C) 1998 by Lucent Technologies All Rights Reserved Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that the copyright notice and this permission notice and warranty disclaimer appear in supporting documentation, and that the name of Lucent or any of its entities not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission. LUCENT DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL LUCENT OR ANY OF ITS ENTITIES BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
Appendix D: Third party acknowledgements

ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT
SHALL LUCENT OR ANY OF ITS ENTITIES BE LIABLE FOR ANY SPECIAL,
INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER
RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION
OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF
OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

hd_init.c License: The author of this software is David M. Gay. Copyright (C) 2000 by
Lucent Technologies All Rights Reserved Permission to use, copy, modify, and
distribute this software and its documentation for any purpose and without fee is hereby
granted, provided that the above copyright notice appear in all copies and that both that
the copyright notice and this permission notice and warranty disclaimer appear in
supporting documentation, and that the name of Lucent or any of its entities not be
used in advertising or publicity pertaining to distribution of the software without specific,
written prior permission. LUCENT DISCLAIMS ALL WARRANTIES WITH REGARD
to this SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL LUCENT OR ANY OF ITS
ENTITIES BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL
DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE,
DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR
OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE
USE OR PERFORMANCE OF THIS SOFTWARE.

misc.c License: The author of this software is David M. Gay. Copyright (C) 1998, 1999 by Lucent Technologies All Rights
Reserved Permission to use, copy, modify, and distribute this software and its
documentation for any purpose and without fee is hereby granted, provided that the
above copyright notice appear in all copies and that both that the copyright notice and
this permission notice and warranty disclaimer appear in supporting documentation,
and that the name of Lucent or any of its entities not be used in advertising or publicity
pertaining to distribution of the software without specific, written prior permission.
LUCENT DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE,
INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS.
IN NO EVENT SHALL LUCENT OR ANY OF ITS ENTITIES BE LIABLE FOR ANY
SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS,
WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS
ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR
PERFORMANCE OF THIS SOFTWARE.

smisc.c License: The author of this software is David M. Gay. Copyright (C) 1998, 1999 by Lucent Technologies All Rights
Reserved Permission to use, copy, modify, and distribute this software and its
documentation for any purpose and without fee is hereby granted, provided that the
above copyright notice appear in all copies and that both that the copyright notice and
this permission notice and warranty disclaimer appear in supporting documentation,
and that the name of Lucent or any of its entities not be used in advertising or publicity
pertaining to distribution of the software without specific, written prior permission.
LUCENT DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE,
INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS.
IN NO EVENT SHALL LUCENT OR ANY OF ITS ENTITIES BE LIABLE FOR ANY
SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS,
WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS
ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR
PERFORMANCE OF THIS SOFTWARE.

strtodg.c License: The author of this software is David M. Gay. Copyright (C) 1998-2001 by Lucent Technologies All Rights Reserved
Permission to use, copy, modify, and distribute this software and its documentation for
any purpose and without fee is hereby granted, provided that the above copyright notice
appear in all copies and that both that the copyright notice and this permission
notice and warranty disclaimer appear in supporting documentation, and that the name
of Lucent or any of its entities not be used in advertising or publicity pertaining to
distribution of the software without specific, written prior permission.

LUCENT DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE,
INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS.
IN NO EVENT SHALL LUCENT OR ANY OF ITS ENTITIES BE LIABLE FOR ANY
SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS,
WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS
ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR
PERFORMANCE OF THIS SOFTWARE. strtord.c License: The author of this software
is David M. Gay. Copyright (C) 1998, 2000 by Lucent Technologies All Rights
Reserved Permission to use, copy, modify, and distribute this software and its
documentation for any purpose and without fee is hereby granted, provided that the
above copyright notice appear in all copies and that both that the copyright notice and
this permission notice and warranty disclaimer appear in supporting documentation,
and that the name of Lucent or any of its entities not be used in advertising or publicity
pertaining to distribution of the software without specific, written prior permission.
LUCENT DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE,
INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS.
IN NO EVENT SHALL LUCENT OR ANY OF ITS ENTITIES BE LIABLE FOR ANY
SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS,
WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS
ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR
PERFORMANCE OF THIS SOFTWARE. sum.c License: The author of this software is
David M. Gay. Copyright (C) 1998 by Lucent Technologies All Rights Reserved
Permission to use, copy, modify, and distribute this software and its documentation for
any purpose and without fee is hereby granted, provided that the above copyright
notice appear in all copies and that both that the copyright notice and this permission
notice and warranty disclaimer appear in supporting documentation, and that the name
of Lucent or any of its entities not be used in advertising or publicity pertaining to
distribution of the software without specific, written prior permission. LUCENT
DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE,
INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS.
IN NO EVENT SHALL LUCENT OR ANY OF ITS ENTITIES BE LIABLE FOR ANY SPECIAL,
INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE. ulp.c License: The author of this software is David M. Gay. Copyright (C) 1998, 1999 by Lucent Technologies All Rights Reserved Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that the copyright notice and this permission notice and warranty disclaimer appear in supporting documentation, and that the name of Lucent or any of its entities not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission. LUCENT DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL LUCENT OR ANY OF ITS ENTITIES BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
Appendix D: Third party acknowledgements

Progress OpenEdge v11.3 may incorporate DOM4J v1.6.1 from MetaStuff, Ltd. Such technology is subject to the following terms and conditions: Copyright 2001-2005 (C) MetaStuff, Ltd. All Rights Reserved. Redistribution and use of this software and associated documentation ("Software"), with or without modification, are permitted provided that the following conditions are met: 1. Redistributions of source code must retain copyright statements and notices. Redistributions must also contain a copy of this document. 2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. 3. The name "DOM4J" must not be used to endorse or promote products derived from this Software without prior written permission of MetaStuff, Ltd. For written permission, please contact dom4j-info@metastuff.com. 4. Products derived from this Software may not be called "DOM4J" nor may "DOM4J" appear in their names without prior written permission of MetaStuff, Ltd. DOM4J is a registered trademark of MetaStuff, Ltd. 5. Due credit should be given to the DOM4J Project - http://www.dom4j.org THIS SOFTWARE IS PROVIDED BY METASTUFF, LTD. AND CONTRIBUTORS "AS IS" AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL METASTUFF, LTD. OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Progress OpenEdge v11.3 may incorporate GraphicsMagick v1.3.14 from GraphicsMagick Group. Such technology is subject to the following terms and conditions:

.. This text is in reStucturedText format, so it may look a bit odd. .. See http://docutils.sourceforge.net/rst.html for details.

======================================
GraphicsMagick Copyrights and Licenses
======================================

This file is part of the GraphicsMagick software distributed by the GraphicsMagick Group.

[*Please note that the legal community considers 15 or more total lines of code or text (not necessarily contiguous) to be significant for the purposes of copyright. Repeated changes such as renaming a symbol has similar significance to changing one line of code.*]

The licenses which components of this software fall under are as follows.

1) In November 2002, the GraphicsMagick Group created GraphicsMagick from ImageMagick Studio's ImageMagick and applied the "MIT" style license: Copyright (C) 2002 - 2012 GraphicsMagick Group
Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions: The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software. THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

2) In October 1999, ImageMagick Studio assumed the responsibility for the development of ImageMagick (forking from the distribution by E. I. du Pont de Nemours and Company) and applied a new license:

Copyright (C) 2002 ImageMagick Studio, a non-profit organization dedicated to making software imaging solutions freely available.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files ("ImageMagick"), to deal in ImageMagick without restriction, including without limitation

the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of ImageMagick, and to permit persons to whom the ImageMagick is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of ImageMagick.

The software is provided "as is", without warranty of any kind, express or implied, including but not limited to the warranties of merchantability, fitness for a particular purpose and noninfringement. In no event shall ImageMagick Studio be liable for any claim, damages or other liability, whether in an action of contract, tort or otherwise, arising from, out of or in connection with ImageMagick or the use or other dealings in ImageMagick.

Except as contained in this notice, the name of the ImageMagick Studio shall not be used in advertising or otherwise to promote the sale, use or other dealings in ImageMagick without prior written authorization from the ImageMagick Studio.

3) From 1991 to October 1999 (through ImageMagick 4.2.9), ImageMagick was developed and distributed by E. I. du Pont de Nemours and Company: Copyright 1999 E. I. du Pont de Nemours and Company

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files ("ImageMagick"), to deal in ImageMagick without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of ImageMagick, and to permit persons to whom the ImageMagick is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of ImageMagick.
The software is provided "as is", without warranty of any kind, express or implied, including but not limited to the warranties of merchantability, fitness for a particular purpose and noninfringement.

In no event shall E. I. du Pont de Nemours and Company be liable for any claim, damages or other liability, whether in an action of contract, tort or otherwise, arising from, out of or in connection with ImageMagick or the use or other dealings in ImageMagick.

Except as contained in this notice, the name of the E. I. du Pont de Nemours and Company shall not be used in advertising or otherwise to promote the sale, use or other dealings in ImageMagick without prior written authorization from the E. I. du Pont de Nemours and Company.

4) The GraphicsMagick Base64Decode() and Base64Encode() functions are based on source code obtained from OpenSSH. This source code is distributed under the following license: Copyright (c) 2000 Markus Friedl. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE AUTHOR "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

5) Many of the pattern images in coders/logo.c are derived from XFig, which is distributed under the following license:

| FIG : Facility for Interactive Generation of figures
| Copyright (c) 1985-1988 by Supoj Sutanthavibul
| Parts Copyright (c) 1989-2000 by Brian V. Smith
| Parts Copyright (c) 1991 by Paul King

Any party obtaining a copy of these files is granted, free of charge, a full and unrestricted irrevocable, world-wide, paid up, royalty-free, nonexclusive right and license to deal in this software and documentation files (the "Software"), including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons who receive copies from any such party to do so, with the only requirement being that this copyright notice remain intact.
6) The documentation for the composition operators is copied from the rlecomp manual page, which is authored by Rod Bogart and John W. Peterson. Rlecomp is part of the Utah Raster Toolkit distributed by the University of Michigan and the University of Utah. The copyright for this manual page is as follows: Copyright (c) 1986, University of Utah

This software is copyrighted as noted below. It may be freely copied, modified, and redistributed, provided that the copyright notice is preserved on all copies.

There is no warranty or other guarantee of fitness for this software, it is provided solely "as is". Bug reports or fixes may be sent to the author, who may or may not act on them as he desires.

You may not include this software in a program or other software product without supplying the source, or without informing the end-user that the source is available for no extra charge.

If you modify this software, you should include a notice giving the name of the person performing the modification, the date of modification, and the reason for such modification.

7) The source code comprising swab.c is originally derived from libtiff which has the following license:

| Copyright (c) 1988-1997 Sam Leffler
| Copyright (c) 1991-1997 Silicon Graphics, Inc.

Permission to use, copy, modify, distribute, and sell this software and its documentation for any purpose is hereby granted without fee, provided that (i) the above copyright notices and this permission notice appear in all copies of the software and related documentation, and (ii) the names of Sam Leffler and Silicon Graphics may not be used in any advertising or publicity relating to the software without the specific, prior written permission of Sam Leffler and Silicon Graphics.

THE SOFTWARE IS PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EXPRESS, IMPLIED OR OTHERWISE, INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. IN NO EVENT SHALL SAM LEFFLER OR SILICON GRAPHICS BE LIABLE FOR ANY SPECIAL, INCIDENTAL, INDIRECT OR CONSEQUENTIAL DAMAGES OF ANY KIND, OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER OR NOT ADVISED OF THE POSSIBILITY OF DAMAGE, AND ON ANY THEORY OF LIABILITY, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

8) The C++ API known as "Magick++", and which resides in the Magick++ directory, is distributed under the following license: Copyright 1999 - 2003 Bob Friesenhahn <bfriesen@simple.dallas.tx.us>

Permission is hereby granted, free of charge, to any person obtaining a copy of the source files and associated documentation files ("Magick++"), to deal in Magick++ without restriction, including without limitation of the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of Magick++, and to permit persons to whom the Magick++ is furnished to do so, subject to the following conditions:

This copyright notice shall be included in all copies or substantial portions of Magick++. The copyright to Magick++ is retained by its author and shall not be subsumed or replaced by any other copyright.
The software is provided "as is", without warranty of any kind, express or implied, including but not limited to the warranties of merchantability, fitness for a particular purpose and noninfringement. In no event shall Bob Friesenhahn be liable for any claim, damages or other liability, whether in an action of contract, tort or otherwise, arising from, out of or in connection with Magick++ or the use or other dealings in Magick++.

9) The GraphicsMagick HaldClutImagePixels() function in magick/hclut.c is based on source code from the HaldCLUT package by Eskil Steenberg (http://www.quelsolaar.com/technology/clut.html) which is distributed under the following license: Copyright (c) 2005 Eskil Steenberg. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE AUTHOR "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

10) GraphicsMagick makes use of third-party "delegate" libraries to support certain optional features. These libraries bear their own copyrights and licenses, which may be more or less restrictive than the GraphicsMagick license. For convenience, when GraphicsMagick is bundled with (or compiled with) "delegate" libraries, a copy of the licenses for these libraries is provided in a "licenses" directory.
This distribution contains the sixth public release of the Independent JPEG Group’s free JPEG software. You are welcome to redistribute this software and to use it for any purpose, subject to the conditions under LEGAL ISSUES, below.

Serious users of this software (particularly those incorporating it into larger programs) should contact IJG at jpeg-info@uunet.uu.net to be added to our electronic mailing list. Mailing list members are notified of updates and have a chance to participate in technical discussions, etc.

This software is the work of Tom Lane, Philip Gladstone, Jim Boucher, Lee Crocker, Julian Minguillon, Luis Ortiz, George Phillips, Davide Rossi, Guido Vollbeding, Ge' Weijers, and other members of the Independent JPEG Group. IJG is not affiliated with the official ISO JPEG standards committee.

OVERVIEW
========

This package contains C software to implement JPEG image compression and decompression. JPEG (pronounced "jay-peg") is a standardized compression method for full-color and gray-scale images. JPEG is intended for compressing "real-world" scenes; line drawings, cartoons and other non-realistic images are not its strong suit. JPEG is lossy, meaning that the output image is not exactly identical to the input image. Hence you must not use JPEG if you have to have identical output bits. However, on typical photographic images, very good compression levels can be obtained with no visible change, and remarkably high compression levels are possible if you can tolerate a low-quality image. For more details, see the references, or just experiment with various compression settings. This software implements JPEG baseline, extended-sequential, and progressive compression processes. Provision is made for supporting all variants of these processes, although some uncommon parameter settings aren't implemented yet.

For legal reasons, we are not distributing code for the arithmetic-coding variants of JPEG; see LEGAL ISSUES. We have made no provision for supporting the hierarchical or lossless processes defined in the standard.

We provide a set of library routines for reading and writing JPEG image files, plus two sample applications "cjpeg" and "djpeg", which use the library to perform conversion between JPEG and some other popular image file formats. The library is intended to be reused in other applications.

In order to support file conversion and viewing software, we have included considerable functionality beyond the bare JPEG coding/decoding capability; for example, the color quantization modules are not strictly part of JPEG decoding, but they are essential for output to colormapped file formats or colormapped displays. These extra functions can be compiled out of the library if not required for a particular application. We have also included "jpegran", a utility for lossless transcoding between different JPEG processes, and "rdjpcgcom" and "wrjpcgcom", two simple applications for inserting and extracting textual comments in JFIF files.

The emphasis in designing this software has been on achieving portability and flexibility, while also making it fast enough to be useful. In particular, the software is not intended to be read as a tutorial on JPEG. (See the REFERENCES section for introductory material.) Rather, it is intended to be reliable, portable, industrial-strength code. We do not claim to have achieved that goal in every aspect of the software, but we strive for it.
Appendix D: Third party acknowledgements

We welcome the use of this software as a component of commercial products. No royalty is required, but we do ask for an acknowledgement in product documentation, as described under LEGAL ISSUES.

LEGAL ISSUES

In plain English:

1. We don't promise that this software works. (But if you find any bugs, please let us know!)

2. You can use this software for whatever you want. You don't have to pay us.

3. You may not pretend that you wrote this software. If you use it in a program, you must acknowledge somewhere in your documentation that you've used the IJG code.

In legalese:

The authors make NO WARRANTY or representation, either express or implied, with respect to this software, its quality, accuracy, merchantability, or fitness for a particular purpose. This software is provided "AS IS", and you, its user, assume the entire risk as to its quality and accuracy.

This software is copyright (C) 1991-1998, Thomas G. Lane. All Rights Reserved except as specified below.

Permission is hereby granted to use, copy, modify, and distribute this software (or portions thereof) for any purpose, without fee, subject to these conditions:

(1) If any part of the source code for this software is distributed, then this README file must be included, with this copyright and no-warranty notice unaltered; and any additions, deletions, or changes to the original files must be clearly indicated in accompanying documentation.

(2) If only executable code is distributed, then the accompanying documentation must state that "this software is based in part on the work of the Independent JPEG Group".

(3) Permission for use of this software is granted only if the user accepts full responsibility for any undesirable consequences; the authors accept NO LIABILITY for damages of any kind.

These conditions apply to any software derived from or based on the IJG code, not just to the unmodified library. If you use our work, you ought to acknowledge us.

Permission is NOT granted for the use of any IJG author's name or company name in advertising or publicity relating to this software or products derived from it. This software may be referred to only as "the Independent JPEG Group's software".

We specifically permit and encourage the use of this software as the basis of commercial products, provided that all warranty or liability claims are assumed by the product vendor.

ansi2knr.c is included in this distribution by permission of L. Peter Deutsch, sole proprietor of its copyright holder, Aladdin Enterprises of Menlo Park, CA.
ansi2knr.c is NOT covered by the above copyright and conditions, but instead by the usual distribution terms of the Free Software Foundation; principally, that you must include source code if you redistribute it. (See the file ansi2knr.c for full details.) However, since ansi2knr.c is not needed as part of any program generated from the IJG code, this does not limit you more than the foregoing paragraphs do.

The Unix configuration script "configure" was produced with GNU Autoconf.

It is copyright by the Free Software Foundation but is freely distributable.

The same holds for its supporting scripts (config.guess, config.sub, ltconfig, ltmain.sh). Another support script, install-sh, is copyright by M.I.T. but is also freely distributable.

It appears that the arithmetic coding option of the JPEG spec is covered by patents owned by IBM, AT&T, and Mitsubishi. Hence arithmetic coding cannot legally be used without obtaining one or more licenses. For this reason, support for arithmetic coding has been removed from the free JPEG software. (Since arithmetic coding provides only a marginal gain over the unpatented Huffman mode, it is unlikely that very many implementations will support it.)

So far as we are aware, there are no patent restrictions on the remaining code.

The IJG distribution formerly included code to read and write GIF files.

To avoid entanglement with the Unisys LZW patent, GIF reading support has been removed altogether, and the GIF writer has been simplified to produce "uncompressed GIFs". This technique does not use the LZW algorithm; the resulting GIF files are larger than usual, but are readable by all standard GIF decoders.

We are required to state that "The Graphics Interchange Format(c) is the Copyright property of CompuServe Incorporated. GIF(sm) is a Service Mark property of CompuServe Incorporated."

Contents of png.txt file (from GraphicsMagick):

This copy of the libpng notices is provided for your convenience. In case of any discrepancy between this copy and the notices in the file png.h that is included in the libpng distribution, the latter shall prevail.

COPYRIGHT NOTICE, DISCLAIMER, and LICENSE:

If you modify libpng you may insert additional notices immediately following this sentence.

libpng versions 1.0.7, July 1, 2000, through 1.2.0, September 1, 2001, are Copyright (c) 2000 Glenn Randers-Pehrson and are distributed according to the same disclaimer and license as libpng-1.0.6 with the following individuals added to the list of Contributing Authors

Simon-Pierre Cadieux

Eric S. Raymond

Gilles Vollant

and with the following additions to the disclaimer:

There is no warranty against interference with your enjoyment of the library or against infringement. There is no warranty that our efforts or the library will fulfill any of your
particular purposes or needs. This library is provided with all faults, and the entire risk of satisfactory quality, performance, accuracy, and effort is with the user.

libpng versions 0.97, January 1998, through 1.0.6, March 20, 2000, are Copyright (c) 1998, 1999 Glenn Randers-Pehrson, and are distributed according to the same disclaimer and license as libpng-0.96, with the following individuals added to the list of Contributing Authors:

  Tom Lane
  Glenn Randers-Pehrson
  Willem van Schaik

libpng versions 0.89, June 1996, through 0.96, May 1997, are Copyright (c) 1996, 1997 Andreas Dilger

Distributed according to the same disclaimer and license as libpng-0.88, with the following individuals added to the list of Contributing Authors:

  John Bowler
  Kevin Bracey
  Sam Bushell
  Magnus Holmgren
  Greg Roelofs
  Tom Tanner

libpng versions 0.5, May 1995, through 0.88, January 1996, are Copyright (c) 1995, 1996 Guy Eric Schalnat, Group 42, Inc. For the purposes of this copyright and license, "Contributing Authors" is defined as the following set of individuals:

  Andreas Dilger
  Dave Martindale
  Guy Eric Schalnat
  Paul Schmidt
  Tim Wegner

The PNG Reference Library is supplied "AS IS". The Contributing Authors and Group 42, Inc. disclaim all warranties, expressed or implied, including, without limitation, the warranties of merchantability and of fitness for any purpose. The Contributing Authors and Group 42, Inc. assume no liability for direct, indirect, incidental, special, exemplary, or consequential damages, which may result from the use of the PNG Reference Library, even if advised of the possibility of such damage. Permission is hereby granted to use, copy, modify, and distribute this source code, or portions hereof, for any purpose, without fee, subject to the following restrictions:

1. The origin of this source code must not be misrepresented.

2. Altered versions must be plainly marked as such and must not be misrepresented as being the original source.
3. This Copyright notice may not be removed or altered from any source or altered source distribution.

The Contributing Authors and Group 42, Inc. specifically permit, without fee, and encourage the use of this source code as a component to supporting the PNG file format in commercial products. If you use this source code in a product, acknowledgment is not required but would be appreciated.

A "png_get_copyright" function is available, for convenient use in "about" boxes and the like:

    printf("%s",png_get_copyright(NULL));

Also, the PNG logo (in PNG format, of course) is supplied in the files "pngbar.png" and "pngbar.jpg (88x31) and "pngnow.png" (98x31).

Libpng is OSI Certified Open Source Software. OSI Certified Open Source is a certification mark of the Open Source Initiative.

Glenn Randers-Pehrson
randeg@alum.rpi.edu
September 1, 2001

Contents of tiff.txt file (from GraphicsMagick):

Copyright (c) 1988-1997 Sam Leffler
Copyright (c) 1991-1997 Silicon Graphics, Inc.

Permission to use, copy, modify, distribute, and sell this software and its documentation for any purpose is hereby granted without fee, provided that (i) the above copyright notices and this permission notice appear in all copies of the software and related documentation, and (ii) the names of Sam Leffler and Silicon Graphics may not be used in any advertising or publicity relating to the software without the specific, prior written permission of Sam Leffler and Silicon Graphics. THE SOFTWARE IS PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND,

EXPRESS, IMPLIED OR OTHERWISE, INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. IN NO EVENT SHALL SAM LEFFLER OR SILICON GRAPHICS BE LIABLE FOR ANY SPECIAL, INCIDENTAL, INDIRECT OR CONSEQUENTIAL DAMAGES OF ANY KIND, OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER OR NOT ADVISED OF THE POSSIBILITY OF DAMAGE, AND ON ANY THEORY OF LIABILITY, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

Contents of zlib.txt file (from GraphicsMagick):

zlib 1.1.3 is a general purpose data compression library. All the code is thread safe. The data format used by the zlib library is described by RFCs (Request for Comments) 1950 to 1952 in the files ftp://ds.internic.net/rfc/rfc1950.txt (zlib format), rfc1951.txt (deflate format) and rfc1952.txt (gzip format). These documents are also available in other formats from ftp://ftp.uu.net/graphics/png/documents/zlib/zdoc-index.html

All functions of the compression library are documented in the file zlib.h (volunteer to write man pages welcome, contact jloup@gzip.org). A usage example of the library is
given in the file example.c which also tests that the library is working correctly. Another example is given in the file minigzip.c. The compression library itself is composed of all source files except example.c and minigzip.c.

To compile all files and run the test program, follow the instructions given at the top of Makefile. In short "make test; make install" should work for most machines. For Unix: "configure; make test; make install"

For MSDOS, use one of the special makefiles such as Makefile.msc.

For VMS, use Make_vms.com or descrip.mms.

Questions about zlib should be sent to <zlib@quest.jpl.nasa.gov>, or to Gilles Vollant <info@winimage.com> for the Windows DLL version.

The zlib home page is http://www.cdrom.com/pub/infozip/zlib/

The official zlib ftp site is ftp://ftp.cdrom.com/pub/infozip/zlib/

Before reporting a problem, please check those sites to verify that you have the latest version of zlib; otherwise get the latest version and check whether the problem still exists or not.

Mark Nelson <markn@tiny.com> wrote an article about zlib for the Jan. 1997 issue of Dr. Dobb's Journal; a copy of the article is available in http://web2.airmail.net/markn/articles/zlibtool/zlibtool.htm

The changes made in version 1.1.3 are documented in the file ChangeLog.

The main changes since 1.1.2 are:

- fix "an inflate input buffer bug that shows up on rare but persistent occasions" (Mark)
- fix gzread and gztell for concatenated .gz files (Didier Le Botlan)
- fix gzseek(..., SEEK_SET) in write mode
- fix crc check after a gzeek (Frank Faubert)
- fix miniunzip when the last entry in a zip file is itself a zip file (J Lilge)
- add contrib/asm586 and contrib/asm686 (Brian Raiter)

See http://www.muppetlabs.com/~breadbox/software/assembly.html

- add support for Delphi 3 in contrib/delphi (Bob Dellaca)
- add support for C++Builder 3 and Delphi 3 in contrib/delphi2 (Davide Moretti)
- do not exit prematurely in untgz if 0 at start of block (Magnus Holmgren)
- use macro EXTERN instead of extern to support DLL for BeOS (Sander Stoks)

- added a FAQ file

plus many changes for portability.
Unsupported third party contributions are provided in directory "contrib". A Java implementation of zlib is available in the Java Development Kit 1.1
http://www.javasoft.com/products/JDK/1.1/docs/api/Package-java.util.zip.html

See the zlib home page http://www.cdrom.com/pub/infozip/zlib/ for details.

A Perl interface to zlib written by Paul Marquess <pmarquess@bfsec.bt.co.uk> is in the CPAN (Comprehensive Perl Archive Network) sites, such as: ftp://ftp.cis.ufl.edu/pub/perl/CPAN/modules/by-module/Compress/Compress-Zlib*

A Python interface to zlib written by A.M. Kuchling <amk@magnet.com> is available in Python 1.5 and later versions, see
http://www.python.org/doc/lib/module-zlib.html

A zlib binding for TCL written by Andreas Kupries <a.kupries@westend.com> is available at http://www.westend.com/~kupries/doc/trf/man/man.html

An experimental package to read and write files in .zip format, written on top of zlib by Gilles Vollant <info@winimage.com>, is available at http://www.winimage.com/zLibDll/unzip.html and also in the contrib/minizip directory of zlib.

Notes for some targets:

- To build a Windows DLL version, include in a DLL project zlib.def, zlib.rc and all .c files except example.c and minigzip.c; compile with -DZLIB_DLL

  The zlib DLL support was initially done by Alessandro Iacopetti and is now maintained by Gilles Vollant <info@winimage.com>. Check the zlib DLL home page at http://www.winimage.com/zLibDll

  From Visual Basic, you can call the DLL functions which do not take a structure as argument: compress, uncompress and all gz* functions.

  See contrib/visual-basic.txt for more information, or get http://www.tcfb.com/dowseware/cmp-z-it.zip

- For 64-bit Irix, deflate.c must be compiled without any optimization. With -O, one libpng test fails. The test works in 32 bit mode (with the -n32 compiler flag). The compiler bug has been reported to SGI.

- zlib doesn't work with gcc 2.6.3 on a DEC 3000/300LX under OSF/1 2.1 it works when compiled with cc.

- on Digital Unix 4.0D (formely OSF/1) on AlphaServer, the cc option -std1 is necessary to get gzprintf working correctly. This is done by configure.

- zlib doesn't work on HP-UX 9.05 with some versions of /bin/cc. It works with other compilers. Use "make test" to check your compiler.

- gzdopen is not supported on RISCOS, BEOS and by some Mac compilers.

- For Turbo C the small model is supported only with reduced performance to avoid any far allocation; it was tested with -DMAX_WBITS=11 -DMAX_MEM_LEVEL=3

- For PalmOs, see http://www.cs.uib.no/~perm/PASTA/pilot/software.html

Per Harald Myrvang <perm@stud.cs.uib.no>

Acknowledgments:
Appendix D: Third party acknowledgements

The deflate format used by zlib was defined by Phil Katz. The deflate and zlib specifications were written by L. Peter Deutsch. Thanks to all the people who reported problems and suggested various improvements in zlib; they are too numerous to cite here.

Copyright notice:

(C) 1995-1998 Jean-loup Gailly and Mark Adler

This software is provided 'as-is', without any express or implied warranty. In no event will the authors be held liable for any damages arising from the use of this software.

Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions:

1. The origin of this software must not be misrepresented; you must not claim that you wrote the original software. If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required.

2. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.

3. This notice may not be removed or altered from any source distribution.

Jean-loup Gailly  Mark Adler
jloup@gzip.org  madler@alumni.caltech.edu

If you use the zlib library in a product, we would appreciate *not* receiving lengthy legal documents to sign. The sources are provided for free but without warranty of any kind. The library has been entirely written by Jean-loup Gailly and Mark Adler; it does not include third-party code.

If you redistribute modified sources, we would appreciate that you include in the file ChangeLog history information documenting your changes.

Progress OpenEdge v11.3 may incorporate International Classes for Unicode (International Components for Unicode) v2.4 from IBM. Such technology is subject to the following terms and conditions: ICU License - The ICU project is licensed under the X License (see also the x.org original), which is compatible with GPL but non-copyleft. The license allows ICU to be incorporated into a wide variety of software projects using the GPL license. The X license is compatible with the GPL, while also allowing ICU to be incorporated into non-open source products. License ICU License - ICU 1.8.1 and later COPYRIGHT AND PERMISSION NOTICE Copyright (c) 1995-2003 International Business Machines Corporation and others All rights reserved. Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, provided that the above copyright notice(s) and this permission notice appear in all copies of the Software and that both the above copyright notice(s) and this permission notice appear in supporting documentation. THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OF THIRD PARTY RIGHTS. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR HOLDERS INCLUDED IN THIS NOTICE BE LIABLE FOR ANY CLAIM, OR ANY SPECIAL INDIRECT OR CONSEQUENTIAL DAMAGES, OR
ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE. Except as contained in this notice, the name of a copyright holder shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Software without prior written authorization of the copyright holder.

All trademarks and registered trademarks mentioned herein are the property of their respective owners.

Progress OpenEdge v11.3 may incorporate International Components for Unicode v4.8.0. Such technology is subject to the following terms and conditions: ICU License - ICU 1.8.1 and later COPYRIGHT AND PERMISSION NOTICE Copyright (c) 1995-2011 International Business Machines Corporation and others All rights reserved.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, provided that the above copyright notice(s) and this permission notice appear in all copies of the Software and that both the above copyright notice(s) and this permission notice appear in supporting documentation.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OF THIRD PARTY RIGHTS. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR HOLDERS INCLUDED IN THIS NOTICE BE LIABLE FOR ANY CLAIM, OR ANY SPECIAL INDIRECT OR CONSEQUENTIAL DAMAGES, OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

Except as contained in this notice, the name of a copyright holder shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Software without prior written authorization of the copyright holder.

All trademarks and registered trademarks mentioned herein are the property of their respective owners.

Progress OpenEdge v11.3 may incorporate java.net args4j v2.0.12 from Kohsuke Kawaguchi. Such technology is subject to the following terms and conditions:
Copyright (c) 2003, Kohsuke Kawaguchi All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Progress OpenEdge v11.3 may incorporate Jaxen v1.1-beta-3 from jaxen.org. Such technology is subject to the following terms and conditions: Project License $Id: LICENSE.txt 1128 2006-02-05 21:49:04Z elharo $ Copyright 2003-2006 The Werken Company. All Rights Reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. Neither the name of the Jaxen Project nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Progress OpenEdge v11.3 may incorporate Jing 20030619 portion of Progress Extensions for Eclipse v2.2.1. Such technology is subject to the following terms and conditions: Jing Copying Conditions. Copyright (c) 2001-2003 Thai Open Source Software Center Ltd. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. Neither the name of the Thai Open Source Software Center Ltd nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
POSSIBILITY OF SUCH DAMAGE. Third-party JARs - This distribution includes some additional JAR files, which have their own copying conditions:

saxon.jar Comes from the Saxon 6.5.2 distribution and is covered by these conditions
xercesImpl.jar xml-apis.jar Come from the Xerces-J 2.4.0 distribution and are covered by the Apache Software License
isorelax.jar Comes from ISO RELAX 2003/01/08 distribution and is covered by the following license: Copyright (c) 2001-2002, SourceForge ISO-RELAX Project (ASAMI Tomoharu, Daisuke Okajima, Kohsuke Kawaguchi, and MURATA Makoto) Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions: The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software. THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Progress OpenEdge v11.3 may incorporate Sun's Jmaki framework v1.0 and Sun's Jmaki framework v1.0.3 from Sun Microsystems, Inc. Such technology is subject to the following terms and conditions: Copyright 1994-2006 Sun Microsystems, Inc. All Rights Reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: Redistribution of source code must retain the above copyright notice, this list of conditions and the following disclaimer. Redistribution in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. Neither the name of Sun Microsystems, Inc. or the names of contributors may be used to endorse or promote products derived from this software without specific prior written permission. This software is provided "AS IS," without a warranty of any kind. ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE HEREBY EXCLUDED. SUN Microsystems, INC. ("SUN") AND ITS LICENSORS SHALL NOT BE LIABLE FOR ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING OR DISTRIBUTING THIS SOFTWARE OR ITS DERIVATIVES. IN NO EVENT WILL SUN OR ITS LICENSORS BE LIABLE FOR ANY LOST REVENUE, PROFIT OR DATA, OR FOR DIRECT, INDIRECT, SPECIAL, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF THE USE OF OR INABILITY TO USE THIS SOFTWARE, EVEN IF SUN HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. You acknowledge that this software is not designed, licensed or intended for use in the design, construction, operation or maintenance of any nuclear facility.

Progress OpenEdge v11.3 may incorporate Jmaki framework v1.8.0 from Sun Microsystems, Inc. Such technology is subject to the following terms and conditions: Copyright 1994-2006 Sun Microsystems, Inc. All Rights Reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: * Redistribution of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
Redistribution in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. Neither the name of Sun Microsystems, Inc. or the names of contributors may be used to endorse or promote products derived from this software without specific prior written permission. This software is provided "AS IS," without a warranty of any kind. ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE HEREBY EXCLUDED. SUN MICROSYSTEMS, INC. ("SUN") AND ITS LICENSORS SHALL NOT BE LIABLE FOR ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING OR DISTRIBUTING THIS SOFTWARE OR ITS DERIVATIVES. IN NO EVENT WILL SUN OR ITS LICENSORS BE LIABLE FOR ANY LOST REVENUE, PROFIT OR DATA, OR FOR DIRECT, INDIRECT, SPECIAL, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF THE USE OF OR INABILITY TO USE THIS SOFTWARE, EVEN IF SUN HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. You acknowledge that this software is not designed, licensed or intended for use in the design, construction, operation or maintenance of any nuclear facility.

Progress OpenEdge v11.3 may incorporate JSTL v1.0 from Sun Microsystems, Inc. Such technologies are subject to the following terms and conditions: Code sample License Copyright 1994-2006 Sun Microsystems, Inc. All Rights Reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: Redistribution of source code must retain the above copyright notice, this list of conditions and the following disclaimer. Redistribution in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. Neither the name of Sun Microsystems, Inc. or the names of contributors may be used to endorse or promote products derived from this software without specific prior written permission. This software is provided "AS IS," without a warranty of any kind. ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE HEREBY EXCLUDED. SUN MICROSYSTEMS, INC. ("SUN") AND ITS LICENSORS SHALL NOT BE LIABLE FOR ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING OR DISTRIBUTING THIS SOFTWARE OR ITS DERIVATIVES. IN NO EVENT WILL SUN OR ITS LICENSORS BE LIABLE FOR ANY LOST REVENUE, PROFIT OR DATA, OR FOR DIRECT, INDIRECT, SPECIAL, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF THE USE OF OR INABILITY TO USE THIS SOFTWARE, EVEN IF SUN HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. You acknowledge that this software is not designed, licensed or intended for use in the design, construction, operation or maintenance of any nuclear facility.

Progress OpenEdge v11.3 may incorporate Looks v2.1.2 from JGoodies. Such technology is subject to the following terms and conditions: The BSD License for the JGoodies Looks - Copyright (c) 2001-2007 JGoodies Karsten Lentzsch. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. Neither the name of JGoodies Karsten Lentzsch nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written
permission. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Progress OpenEdge v11.3 may incorporate OpenSSL v0.9.8 from The OpenSSL Project. Such technology is subject to the following terms and conditions: OpenSSL License. Copyright (c) 1998-2006 The OpenSSL Project. All rights reserved.
Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: 1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. 2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. 3. All advertising materials mentioning features or use of this software must display the following acknowledgment: "This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit. (http://www.openssl.org/)." 4. The names "OpenSSL Toolkit" and "OpenSSL Project" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact openssl-core@openssl.org. 5. Products derived from this software may not be called "OpenSSL" nor may "OpenSSL" appear in their names without prior written permission of the OpenSSL Project. 6. Redistributions of any form whatsoever must retain the following acknowledgment: "This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit (http://www.openssl.org/)." THIS SOFTWARE IS PROVIDED BY THE OpenSSL PROJECT "AS IS" AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE OpenSSL PROJECT OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

OpenSSL includes cryptographic software written by Eric Young (eay@cryptsoft.com). This product includes software written by Tim Hudson (tjh@cryptsoft.com). Original SSLeay License. Copyright (C) 1995-1998 Eric Young (eay@cryptsoft.com) All rights reserved. This package is an SSL implementation written by Eric Young (eay@cryptsoft.com). The implementation was written so as to conform with Netscapes SSL. This library is free for commercial and non-commercial use as long as the following conditions are aheared to. The following conditions apply to all code found in this distribution, be it the RC4, RSA, Ihash, DES, etc., code; not just the SSL code. The SSL documentation included with this distribution is covered by the same copyright terms except that the holder is Tim Hudson (tjh@cryptsoft.com). Copyright remains Eric Young's, and as such any Copyright notices in the code are not to be removed. If this package is used in a product, Eric Young should be given attribution as the author
of the parts of the library used. This can be in the form of a textual message at program startup or in documentation (online or textual) provided with the package.
Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: 1. Redistributions of source code must retain the copyright notice, this list of conditions and the following disclaimer. 2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. 3. All advertising materials mentioning features or use of this software must display the following acknowledgement: "This product includes cryptographic software written by Eric Young (eay@cryptsoft.com)." The word 'cryptographic' can be left out if the routines from the library being used are not cryptographic related :-). 4. If you include any Windows specific code (or a derivative thereof) from the apps directory (application code) you must include an acknowledgement: "This product includes software written by Tim Hudson (tjh@cryptsoft.com)." THIS SOFTWARE IS PROVIDED BY ERIC YOUNG ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. The licence and distribution terms for any publically available version or derivative of this code cannot be changed. i.e. this code cannot simply be copied and put under another distribution licence [including the GNU Public Licence.]

Progress OpenEdge v11.3 may incorporate Quartz Enterprise Job Scheduler v1.3.2 from James House. Such technologies are subject to the following terms and conditions: Copyright James House (c) 2001-2003 All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: 1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. 2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION). HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. This product uses and includes within its distribution, software developed by the Apache Software Foundation (http://www.apache.org/).

Progress OpenEdge v11.3 may incorporate relaxngDatatype.jar portion of Progress Extensions for Eclipse v2.2. Such technology is subject to the following terms and conditions: Copyright (c) 2001, Thai Open Source Software Center Ltd, Sun Microsystems. All rights reserved. Redistribution and use in source and binary forms,
with or without modification, are permitted provided that the following conditions are met: Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. Neither the names of the copyright holders nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Progress OpenEdge v11.3 may incorporate SLF4J v1.6.2 and SLF4J Log4J v1.6.1 from slf4j.org. Such technology is subject to the following terms and conditions: Copyright (c) 2004-2008 QOS.ch All rights reserved. Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions: The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software. THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Progress OpenEdge v11.3 may incorporate STAX API (JSR 173) v3.1.1 from Woodstox Project. Such technology is subject to the following terms and conditions: Copyright (c) 2004-2010, Woodstox Project (http://woodstox.codehaus.org/) All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3. Neither the name of the Woodstox XML Processor nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.
THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Progress OpenEdge v11.3 may incorporate Trang 20030619 portion of Progress Extensions for Eclipse v2.2.1. Such technology is subject to the following terms and conditions: Copyright (c) 2002, 2003 Thai Open Source Software Center Ltd. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. Neither the name of the Thai Open Source Software Center Ltd nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Progress OpenEdge v11.3 may incorporate xpp3-1.1.3.4.O portion of Progress Extensions for Eclipse v2.2.1. Such technology is subject to the following terms and conditions: Indiana University Extreme! Lab Software License Version 1.1.1 Copyright (c) 2002 Extreme! Lab, Indiana University. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3. The end-user documentation included with the redistribution, if any, must include the following acknowledgment:

"This product includes software developed by the Indiana University Extreme! Lab (http://www.extreme.indiana.edu/)."
Alternately, this acknowledgment may appear in the software itself, if and wherever such third-party acknowledgments normally appear.

4. The names "Indiana University" and "Indiana University Extreme! Lab" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact http://www.extreme.indiana.edu/.

5. Products derived from this software may not use "Indiana University" name nor may "Indiana University" appear in their name, without prior written permission of the Indiana University.

THIS SOFTWARE IS PROVIDED "AS IS" AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHORS, COPYRIGHT HOLDERS OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Progress OpenEdge v11.3 may incorporate Xpp3 v1.1.3.4 from Indiana University. Such technology is subject to the following terms and conditions: Indiana University Extreme! Lab Software License Version 1.1.1 Copyright (c) 2002 Extreme! Lab, Indiana University. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: 1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. 2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. 3. The end-user documentation included with the redistribution, if any, must include the following acknowledgment: "This product includes software developed by the Indiana University Extreme! Lab (http://www.extreme.indiana.edu/)." Alternately, this acknowledgment may appear in the software itself, if and wherever such third-party acknowledgments normally appear. 4. The names "Indiana University" and "Indiana University Extreme! Lab" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact http://www.extreme.indiana.edu/. 5. Products derived from this software may not use "Indiana University" name nor may "Indiana University" appear in their name, without prior written permission of the Indiana University. THIS SOFTWARE IS PROVIDED "AS IS" AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHORS, COPYRIGHT HOLDERS OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
Progress OpenEdge v11.3 may incorporate XStream v1.3.1 from XStream. Such technology is subject to the following terms and conditions: Copyright (c) 2003-2006, Joe Walnes Copyright (c) 2006-2007, XStream Committers All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. Neither the name of XStream nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Progress OpenEdge v11.3 may incorporate YAJL v0.4.0 from Lloyd Hilaiel. Such technology is subject to the following terms and conditions: Copyright 2007, Lloyd Hilaiel. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: 1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. 2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. 3. Neither the name of Lloyd Hilaiel nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission. THIS SOFTWARE IS PROVIDED BY THE AUTHOR "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Progress OpenEdge v11.3 may incorporate GNU ZLIB v1.1.4 and GNU ZLIB v1.2.3 from Jean-loup Gailly & Mark Alder. Such technology is subject to the following terms and conditions: Copyright notice: (C) 1995-2002 Jean-loup Gailly and Mark Adler This software is provided 'as-is', without any express or implied warranty. Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions: 1. The origin of this software must not be misrepresented; you must not claim that you wrote the original software. If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required. 2. Altered source versions must be plainly marked as such, and must not be
misrepresented as being the original software. 3. This notice may not be removed or altered from any source distribution.

Jean-loup Gailly       Mark Adler
jloup@gzip.org          madler@alumni.caltech.edu

Progress OpenEdge v11.3 may incorporate ZLIB.NET Free v1.0.4 from ComponentAce. Such technology is subject to the following terms and conditions: Copyright (c) 2006-2007, ComponentAce http://www.componentace.com All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. Neither the name of ComponentAce nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Progress OpenEdge v11.3 may incorporate LALR Parser Generator in Java v0.10k. Such technology is subject to the following terms and conditions: CUP Parser Generator Copyright Notice, License, and Disclaimer – Copyright 1996-1999 by Scott Hudson, Frank Flannery, C. Scott Ananian - Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both the copyright notice and this permission notice and warranty disclaimer appear in supporting documentation, and that the names of the authors or their employers not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission. The authors and their employers disclaim all warranties with regard to this software, including all implied warranties of merchantability and fitness. In no event shall the authors or their employers be liable for any special, indirect or consequential damages or any damages whatsoever resulting from loss of use, data or profits, whether in an action of contract, negligence or other tortious action, arising out of or in connection with the use or performance of this software.

Progress OpenEdge v11.3 includes the RSA Data Security, Inc. MD5 Message-Digest Algorithm. Copyright ©1991-2, RSA Data Security, Inc. Created 1991. All rights reserved. (MD5 Encryption Library v3.0 and MD5 Encryption vMD5C.C) These technologies are subject to the following terms and conditions: RSA Data Security MD5 message-digest algorithm RSA Data Security, Inc. MD5C.C - RSA Data Security, Inc., MD5 message-digest algorithm Copyright (C) 1991-2, RSA Data Security, Inc. Created 1991. All rights reserved. License to copy and use this software is granted provided that it is identified as the "RSA Data Security, Inc. MD5 Message-Digest Algorithm" in all material mentioning or referencing this software or this function. License is also granted to make and use derivative works provided that such works are identified as "derived from the RSA Data Security, Inc. MD5 Message-Digest
Algorithm® in all material mentioning or referencing the derived work. RSA Data Security, Inc. makes no representations concerning either the merchantability of this software or the suitability of this software for any particular purpose. It is provided "as is" without express or implied warranty of any kind. These notices must be retained in any copies of any part of this documentation and/or software.