**Contents**

**Chapter 1: Operational Insight Programmer’s Guide Overview**........................................... 1  
  Operational Insight Architecture ............................................................................................ 1  
  Operational Insight Help ....................................................................................................... 3  
  Conventions Used in This Book ............................................................................................ 3  
  Honeywell Product Support .................................................................................................. 4  
  Contacting Us .................................................................................................................... 5  

**Chapter 2: Basic HTML and Scripting**....................................................................................... 7  
  Creating HTML Pages .......................................................................................................... 7  
  Basic HTML Elements ........................................................................................................... 7  
  *HTML Examples* ................................................................................................................ 8  
  Common HTML Elements ..................................................................................................... 9  
  Advanced HTML Elements ................................................................................................. 11  
  Dynamic HTML ................................................................................................................... 12  
  Styles ................................................................................................................................... 12  
  Using PNSP to Generate HTML ........................................................................................... 13  
  Band Element ...................................................................................................................... 14  
  DataSet Element .................................................................................................................. 15  
  ErrorText and Error Elements ............................................................................................. 17  
  Global Element .................................................................................................................... 18  
  Import Element .................................................................................................................... 18  
  Login Element ..................................................................................................................... 19  
  Permission Element ............................................................................................................. 19  
  Value Element ..................................................................................................................... 20  
  Variable Element .................................................................................................................. 22  
  Using Parameters with PNSP ............................................................................................... 22  
  Scripting ............................................................................................................................... 22  
  VBScript vs. JavaScript ......................................................................................................... 23  
  Creating Scripts in HTML Pages ........................................................................................... 23  
  Accessing HTML Objects from Script ................................................................................... 24  
  Handling HTML Events ........................................................................................................ 24  

**Chapter 3: NetPortal Data Access Interface**........................................................................... 27  
  URL Queries ........................................................................................................................ 27  
  URL Query Syntax ................................................................................................................. 27  
  Format Specifiers .................................................................................................................. 27  
  Data Access Commands ....................................................................................................... 28  
  SQL-based Data Access ........................................................................................................ 28  
    *SQL Command Syntax* .................................................................................................... 28  
    Writing BLOB Data ............................................................................................................ 29  
  Tag-based Data Access ........................................................................................................ 30  
    *Real-time Data Queries* .................................................................................................. 31  
    *Historical Data Queries* ................................................................................................. 32  
  Advanced Data Access Commands ...................................................................................... 33  
    Using rawTagHistory ........................................................................................................ 34  
    Using processedTagHistory ........................................................................................... 35  
  Abstracting Data Sources .................................................................................................... 36  
    Using SQL to Retrieve Tag Data ....................................................................................... 36  
    Using Tag-Based Commands to Get Relational Data ....................................................... 37  
  Metadata Access Commands ............................................................................................... 37
### Chapter 6: Operational Insight Messages and Dialog Boxes

<table>
<thead>
<tr>
<th>Message Type</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Messages</td>
<td>75</td>
</tr>
<tr>
<td>MessageDialog</td>
<td>75</td>
</tr>
<tr>
<td>AlertDialog</td>
<td>76</td>
</tr>
<tr>
<td>PromptUser</td>
<td>77</td>
</tr>
<tr>
<td>ConfirmDialog</td>
<td>78</td>
</tr>
<tr>
<td>ErrorMessage</td>
<td>78</td>
</tr>
<tr>
<td>ExtendedAlertDialog</td>
<td>78</td>
</tr>
</tbody>
</table>

### Application Dialog Boxes

<table>
<thead>
<tr>
<th>Dialog Type</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Save</td>
<td>79</td>
</tr>
<tr>
<td>Load</td>
<td>81</td>
</tr>
<tr>
<td>Color</td>
<td>83</td>
</tr>
<tr>
<td>Font</td>
<td>83</td>
</tr>
<tr>
<td>Tag Search</td>
<td>84</td>
</tr>
<tr>
<td>SQL Editor</td>
<td>87</td>
</tr>
<tr>
<td>DateTime</td>
<td>88</td>
</tr>
</tbody>
</table>

### Chapter 7: Operational Insight Global Functions

<table>
<thead>
<tr>
<th>Global Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Numerical</td>
<td>91</td>
</tr>
<tr>
<td>Date/Time</td>
<td>92</td>
</tr>
<tr>
<td>Global Variables</td>
<td>93</td>
</tr>
</tbody>
</table>

### Chapter 8: Using the WebDataSet

<table>
<thead>
<tr>
<th>WebDataSet Object</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Including the Applet</td>
<td>95</td>
</tr>
<tr>
<td>Applet Parameters</td>
<td>95</td>
</tr>
<tr>
<td>WebDataSet Objects</td>
<td>96</td>
</tr>
<tr>
<td>JWebDataSet Object</td>
<td>96</td>
</tr>
<tr>
<td>JWebRecordSet Object</td>
<td>98</td>
</tr>
<tr>
<td>XField Object</td>
<td>98</td>
</tr>
</tbody>
</table>

### Accessing Metadata

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
</tr>
</tbody>
</table>

### Chapter 9: NetTrend Programming Interface

<table>
<thead>
<tr>
<th>Interface</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Including the Applet</td>
<td>103</td>
</tr>
<tr>
<td>Applet Parameters</td>
<td>103</td>
</tr>
<tr>
<td>Configuring Trends Programmatically</td>
<td>105</td>
</tr>
<tr>
<td>Adding a Line Series</td>
<td>106</td>
</tr>
<tr>
<td>PNChartSeries Object</td>
<td>107</td>
</tr>
<tr>
<td>PNDatasourceManager Object</td>
<td>110</td>
</tr>
<tr>
<td>Adding Functions</td>
<td>111</td>
</tr>
<tr>
<td>Creating Custom Calculations</td>
<td>112</td>
</tr>
<tr>
<td>Adding Constants</td>
<td>114</td>
</tr>
<tr>
<td>Adding Fill Plots</td>
<td>116</td>
</tr>
<tr>
<td>Adding Bars</td>
<td>117</td>
</tr>
<tr>
<td>Adding Axes</td>
<td>118</td>
</tr>
<tr>
<td>Trend Events</td>
<td>120</td>
</tr>
<tr>
<td>OnClick</td>
<td>120</td>
</tr>
<tr>
<td>OnSeriesClick</td>
<td>121</td>
</tr>
<tr>
<td>OnDoubleClick</td>
<td>122</td>
</tr>
</tbody>
</table>
Chapter 10: Other Operational Insight Objects ................................................................. 129
  DataHub Object ............................................................................................................. 129
  Menu Applet .................................................................................................................. 131
  Calendar Control .......................................................................................................... 133
  Trend Control Bar ......................................................................................................... 134
  Toolbar Object .............................................................................................................. 136
  ConfigFiler Object ........................................................................................................ 139
  Tabbed Dialog Box Control ........................................................................................... 140
  Group Box Control ....................................................................................................... 142
  List Control .................................................................................................................. 143
  Slider Control .............................................................................................................. 146
  Session Manager .......................................................................................................... 147

Chapter 11: NetDAC Server-side Data Access ................................................................. 149
  Connecting to the NetServer ........................................................................................ 149
    Explicitly Logging On .................................................................................................. 149
  NetDAC RecordSet Object ............................................................................................ 151
  Writing BLOB Data Using NetDAC .............................................................................. 152
    Using WriteStringBlob ............................................................................................. 152
    Using WriteVariantBlob ......................................................................................... 153
  Writing Tag Values Using NetDAC ............................................................................ 153
    Using WriteTag ........................................................................................................ 153
    Using WriteTags ...................................................................................................... 154

Chapter 12: Using ADO to Get Operational Insight Data ............................................... 157
  Excel and ADO .............................................................................................................. 157
    Using a Recordset Object ......................................................................................... 157
    Using MSXML .......................................................................................................... 158

Chapter 13: Creating Packet Providers .......................................................................... 159
  Obtaining a Provider ...................................................................................................... 159
  Packet Provider Interface ............................................................................................. 159
    Method Details .......................................................................................................... 160
  Sample Packet Provider ............................................................................................... 161
    Creating the Project ................................................................................................... 161
    Sample Code ............................................................................................................ 162

Chapter 14: NetDraw Software Development Kit .......................................................... 165
  NetDraw GUI Configuration ......................................................................................... 165
Appendix A: NetObjects Stored Procedures .............................................................. 207

NetObjects Stored Procedures .................................................................................. 207
User Subsystem........................................................................................................... 207
  dp_addUser .............................................................................................................. 207
  dp_removeUser ................................................................................................. 208
  dp_editUserInfo ................................................................................................. 208
  dp_getUserInfo ...................................................................................................... 208
Relationship Type Subsystem...................................................................................... 209
  dp_addRelationshipType .................................................................................... 209
  dp_removeRelationshipType .............................................................................. 209
  dp_editRelationshipType .................................................................................... 210
  dp_getRelationshipTypeInfo ................................................................................. 210
Image Subsystem...................................................................................................... 211
  dp_addImage ........................................................................................................ 211
  dp_removeImage .................................................................................................. 211
  dp_editImageInfo ................................................................................................. 211
  dp_getImageInfo .................................................................................................. 212
Module Subsystem..................................................................................................... 212
  dp_addModule ..................................................................................................... 212
  dp_removeModule ............................................................................................... 213
  dp_getModuleInfo ................................................................................................ 213
  dp_editModuleInfo .............................................................................................. 213
  dp_addModuleView ............................................................................................... 214
  dp_removeModuleView ....................................................................................... 214
  dp_editModuleView .............................................................................................. 214
  dp_getModuleViewInfo ........................................................................................ 215
  dp_addViewCollectionAssoc ............................................................................... 216
  dp_removeViewCollectionAssoc .......................................................................... 216
  dp_getViewCollectionAssoc ............................................................................... 216
  dp_addModuleMethod ........................................................................................... 217
  dp_removeModuleMethod ..................................................................................... 217
  dp_editModuleMethod .......................................................................................... 218
  dp_getModuleMethodInfo ...................................................................................... 218
  dp_addModuleCollection ...................................................................................... 219
  dp_removeModuleCollection ............................................................................... 219
  dp_editModuleCollection ..................................................................................... 220
  dp_getModuleCollectionInfo ............................................................................... 220
  dp_getCollObjectTypes ......................................................................................... 221
  dp_getCollAttrGroups ......................................................................................... 221
  dp_addMethodCollectionAssoc ......................................................................... 222
  dp_removeMethodCollectionAssoc .................................................................... 222
Attribute Group Subsystem ............................................................................................................ 235

dp_getAttrStateSubscriptionInfo .......................................................... 233
dp_removeAttrStateSubscription ........................................................... 233

dp_addAttrSubscription ........................................................................... 229

dp_getOverrideValue .................................................................................. 228

dp_addAttrGroupCollAssoc ........................................................................ 240

dp_listRelAuditAttrs ................................................................................ 239

dp_removeRelAttrFromAudit ................................................................. 239

dp_addRelAttrToAudit ................................................................................ 238

dp_getAttrStateInfo .................................................................................. 231

dp_removeAttrState .................................................................................. 231

dp_getAttrStateInfoByValue ..................................................................... 231

dp_addAttrStateSubscription .................................................................... 229

dp_removeAttrStateSubscription ............................................................ 233

dp_getAttrStateSubscriptionInfo ............................................................ 233

dp_setAttributeConstraint ....................................................................... 234

dp_getAttributeConstraint ....................................................................... 234

Attribute Subsystem ........................................................................................... 224

dp_addAttribute ......................................................................................... 224

dp_removeAttribute .................................................................................... 225

dp_setDefaultValue ..................................................................................... 225

dp_getDefaultValue ..................................................................................... 225

dp_editAttribute ......................................................................................... 226

dp_getAttributeInfo ................................................................................... 226

dp_overrideDefaultValue ............................................................................ 227

dp_getOverrideValue .................................................................................. 228

dp_addAttrSubscription ............................................................................. 228

dp_removeAttrSubscription ....................................................................... 229

dp_getAttrSubscriptionInfo ....................................................................... 229

dp_addAttrState ......................................................................................... 230

dp_editAttrState ......................................................................................... 230

dp_removeAttrState .................................................................................... 230

dp_getAttrStateInfo ................................................................................... 231

dp_getAttrStateInfoByValue ..................................................................... 231

dp_addAttrStateSubscription .................................................................... 229

dp_removeAttrStateSubscription ............................................................ 233

dp_getAttrStateSubscriptionInfo ............................................................ 233

dp_setAttributeConstraint ....................................................................... 234

dp_getAttributeConstraint ....................................................................... 234

Object Type Subsystem .................................................................................. 241

dp_addToObjectType .................................................................................... 241

dp_removeToObjectType ............................................................................. 242

dp_editToObjectType ................................................................................... 242

dp_getToObjectTypeInfo .............................................................................. 243

dp_addToObjectTypeToRelation ............................................................... 243
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>dp_removeObjectTypeFromRelation</td>
<td>244</td>
</tr>
<tr>
<td>dp_getObjectTypesByRelation</td>
<td>244</td>
</tr>
<tr>
<td>dp_getObjectTypeRelationTree</td>
<td>245</td>
</tr>
<tr>
<td>dp_addObjTypeAttrGroupAssoc</td>
<td>246</td>
</tr>
<tr>
<td>dp_removeObjTypeAttrGroupAssoc</td>
<td>246</td>
</tr>
<tr>
<td>dp_getObjTypeAttrGroupAssoc</td>
<td>247</td>
</tr>
<tr>
<td>dp_addObjTypeViewAssoc</td>
<td>247</td>
</tr>
<tr>
<td>dp_removeObjTypeViewAssoc</td>
<td>248</td>
</tr>
<tr>
<td>dp_setObjTypeDefaultView</td>
<td>248</td>
</tr>
<tr>
<td>dp_clearObjTypeDefaultView</td>
<td>248</td>
</tr>
<tr>
<td>dp_getObjTypeViewAssoc</td>
<td>248</td>
</tr>
<tr>
<td>dp_addObjTypeViewAssoc</td>
<td>248</td>
</tr>
<tr>
<td>dp_removeObjTypeViewAssoc</td>
<td>249</td>
</tr>
<tr>
<td>dp_setObjTypeDefaultView</td>
<td>249</td>
</tr>
<tr>
<td>dp_clearObjTypeDefaultView</td>
<td>249</td>
</tr>
<tr>
<td>dp_getObjTypeViewAssoc</td>
<td>249</td>
</tr>
<tr>
<td>dp_addObjTypeViewAssoc</td>
<td>249</td>
</tr>
<tr>
<td>dp_removeObjTypeViewAssoc</td>
<td>249</td>
</tr>
<tr>
<td>dp_getObjectInfo</td>
<td>251</td>
</tr>
<tr>
<td>dp_editObjectInfo</td>
<td>251</td>
</tr>
<tr>
<td>dp_listObjectAttrGroups</td>
<td>251</td>
</tr>
<tr>
<td>dp_addObjectToRelation</td>
<td>252</td>
</tr>
<tr>
<td>dp_removeObjectFromRelation</td>
<td>252</td>
</tr>
<tr>
<td>dp_getObjectsByRelation</td>
<td>253</td>
</tr>
<tr>
<td>dp_getObjectRelationTree</td>
<td>254</td>
</tr>
<tr>
<td>dp_getObjectTree</td>
<td>255</td>
</tr>
<tr>
<td>dp_setAttrGroupPerms</td>
<td>256</td>
</tr>
<tr>
<td>dp_removeAttrGroupPerms</td>
<td>256</td>
</tr>
<tr>
<td>dp_getAttrGroupPerms</td>
<td>257</td>
</tr>
<tr>
<td>dp_setObjectUserPerms</td>
<td>258</td>
</tr>
<tr>
<td>dp_getObjectUserPerms</td>
<td>258</td>
</tr>
<tr>
<td>dp_removeObjectUserPerms</td>
<td>259</td>
</tr>
<tr>
<td>dp_getAllData</td>
<td>259</td>
</tr>
<tr>
<td>dp_getAttributeData</td>
<td>260</td>
</tr>
<tr>
<td>dp_setAttributeData</td>
<td>261</td>
</tr>
</tbody>
</table>

**Object Subsystem**

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>dp_addObject</td>
<td>250</td>
</tr>
<tr>
<td>dp_removeObject</td>
<td>250</td>
</tr>
<tr>
<td>dp_getObjectInfo</td>
<td>251</td>
</tr>
<tr>
<td>dp_editObjectInfo</td>
<td>251</td>
</tr>
<tr>
<td>dp_listObjectAttrGroups</td>
<td>251</td>
</tr>
<tr>
<td>dp_addObjectToRelation</td>
<td>252</td>
</tr>
<tr>
<td>dp_removeObjectFromRelation</td>
<td>252</td>
</tr>
<tr>
<td>dp_getObjectsByRelation</td>
<td>253</td>
</tr>
<tr>
<td>dp_getObjectRelationTree</td>
<td>254</td>
</tr>
<tr>
<td>dp_getObjectTree</td>
<td>255</td>
</tr>
<tr>
<td>dp_setAttrGroupPerms</td>
<td>256</td>
</tr>
<tr>
<td>dp_removeAttrGroupPerms</td>
<td>256</td>
</tr>
<tr>
<td>dp_getAttrGroupPerms</td>
<td>257</td>
</tr>
<tr>
<td>dp_setObjectUserPerms</td>
<td>258</td>
</tr>
<tr>
<td>dp_getObjectUserPerms</td>
<td>258</td>
</tr>
<tr>
<td>dp_removeObjectUserPerms</td>
<td>259</td>
</tr>
<tr>
<td>dp_getAllData</td>
<td>259</td>
</tr>
<tr>
<td>dp_getAttributeData</td>
<td>260</td>
</tr>
<tr>
<td>dp_setAttributeData</td>
<td>261</td>
</tr>
</tbody>
</table>

**User Accessible Database Views**

<table>
<thead>
<tr>
<th>View</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>V_OBJECTS</td>
<td>262</td>
</tr>
<tr>
<td>V_OBJECT_VALUES</td>
<td>262</td>
</tr>
<tr>
<td>V_OBJECT_RELATIONSHIPS</td>
<td>263</td>
</tr>
<tr>
<td>V_AUDIT_TRAIL</td>
<td>263</td>
</tr>
</tbody>
</table>
Chapter 1: Operational Insight
Programmer’s Guide Overview

This guide is designed to give you the skills required to build Web-based client and server applications using Operational Insight’s tools and technologies. The guide assumes you have:

• Some programming knowledge.
• An understanding of the concepts of objects, methods, properties, and functions.
• A working knowledge of the Operational Insight user interface and capabilities.

Operational Insight Architecture

Operational Insight is used to visualize process and relational data in a Web browser. Operational Insight is also:

• A data access infrastructure.
• A Web portal.
• A platform for building client–server applications.

Operational Insight applications typically run as thin-client applications within a Web browser.

Note

Thin client means that software does not need to be installed on the client machine in order to use Operational Insight.
Operational Insight uses a modular design that takes advantage of the operating system and Web server’s scalability. The following diagram illustrates the major components of Operational Insight.

Operational Insight Components

- **NetPortal**: The main entry point for all data requests. It is referenced heavily when developing applications for Operational Insight.
- **Packet Providers**: Used by the NetPortal to format data. Standard formats are HTML, Excel, CSV, and XML. You can create new packet providers using VB, C++, and other programming languages.
- **PNSP**: A markup language used to access data from Operational Insight and format it for HTML pages.
- **NetServer and Drivers**: The NetServer and its drivers (such as ODBC, OPC, and PI) are responsible for retrieving data from the source systems for the NetPortal.
- **NetDAC**: Used on the server to access data from NetServer for use in ASP, PHP, and other Web scripting languages.
Operational Insight Help

Conventions Used in This Book

This book uses the following conventions.

Conventions Used

<table>
<thead>
<tr>
<th>Convention</th>
<th>Used To</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Bold</strong></td>
<td>Highlight items that belong to the interface, such as buttons, menus, check boxes, and so on.</td>
</tr>
<tr>
<td><em>Italic</em></td>
<td>Highlight book titles and items that require emphasis.</td>
</tr>
<tr>
<td><strong>Courier</strong></td>
<td>Highlight filenames, functions, code examples, or text that a user must type as input.</td>
</tr>
<tr>
<td><strong>LOW CAPS</strong></td>
<td>Highlight keys to be pressed on the keyboard.</td>
</tr>
</tbody>
</table>

Programming code samples appear in fixed font and are indented:

```
variable1 = 2 + variable2
```

Optional code or text appears within square brackets:

[Optional]

In addition to the conventions above, you will also find **Note**, **FMI**, **Tip**, and **Caution** messages throughout the manual. These messages contain valuable information that is important for you to read and understand. Although this book may not contain all message types, you may encounter one or more of the four types:

---

**Note**

Directs your attention to information on memory limitations, specific configurations, or operating systems.

---

**FMI**

(For More Information) Directs you to additional resources.
Tip
Suggests an alternative method but is not essential to use the product.

Caution
Informs you of possible data loss or program failure and provides preventative actions.

Honeywell Product Support
Honeywell's Product Support Specialists are trained service professionals committed to providing the highest quality technical support. Honeywell's Product Support service provides the following:
• Help with product issues involving operation and functionality.
• Problem isolation and identification.
• Problem resolution and workarounds.
• Error message analysis and resolution.
• Up-to-date information on current releases, product compatibility, restrictions, enhancements, and fixes.
• Unlimited access to Honeywell's online Product Knowledge Base.
Contacting Us

Customer Care Center Listing

North America
Email: asksc@Honeywell.com
Canada and United States
Phone: 800-822-7673
Phone: 602-283-1866

Middle East and South Africa
Email: HPSCustomerSupport@honeywell.com
Bahrain - Phone: 8008 1343
Oman - Phone: 8007 7595
Qatar - Phone: 800 5460
Saudi Arabia - Phone: 800 844 5309
South Africa - Phone: 0800 983 634
Turkey - Phone: 00800 448823587
United Arab Emirates - Phone: 8000 444 0300

Europe
Email: HPSCustomerSupport@honeywell.com
Austria - Phone: 0800 00843
Belgium - Phone: 0800 485 80
Bulgaria - Phone: 700 20771
Croatia - Phone: 0800 80 6392
Czech Republic - Phone: 800 142 784
Denmark - Phone: 80 25 21 85
Finland - Phone: 0800 915 938
France - Phone: 0805 10 00 41
Germany - Phone: 0800 7239098
Greece - Phone: 00800 12 9493
Hungary - Phone: 06 800 20 699
Ireland - Phone: 1800 938 488
Italy - Phone: 8000 36205
Israel - Phone: 1 809 407 309
Luxembourg - Phone: 8002 8524
Netherlands - Phone: 0900 0203498
Norway - Phone: 890 11 479
Poland - Phone: 00 800 121 50 46
Portugal - Phone: 800 655 994
Romania - Phone: 0 800 800 178
Russia Federation - Phone: 8 10 80 02-412 50 11
Slovakia - Phone: 0800 002 340
Spain - Phone: 800 099 804
Sweden - Phone: 020 088 3187
Switzerland - Phone: 0800 080 035
United Kingdom - Phone: 0800 279 7226

Asia Pacific
Coming Soon!
Advanced Solutions TAC

North America
Canada and United States - askssc@Honeywell.com
Phone: 800-822-7673
Phone: 602-293-1966

EMEA
Phone: 32 2 728 2200

Asia Pacific
Australia - GTAC@honeywell.com
Phone: 1300-300-4822 (toll free within Australia)
+61-8-9362-9559 (outside Australia)

India - Global-TAC-India@honeywell.com
Phone: +91-20- 66039400

Korea - Global-TAC-Korea@honeywell.com
Phone: +82-2-799-6317
+82-11-9227-6324

China - Global-TAC-China@honeywell.com
Phone: +86-21-52574568

Singapore - GTAC-SEA@honeywell.com
Phone: +65-6580-3500

Taiwan - Global-TAC-Taiwan@honeywell.com
Phone: +886-7-536 2567

Japan - Global-TAC-Japan JA25@honeywell.com
Phone: +81-3-5730-7160
Chapter 2: Basic HTML and Scripting

To create client-side applications in Operational Insight, you need to know HTML and either JavaScript or VBScript. The following sections provide enough information to complete the remainder of this guide, but offer only a very basic introduction to these technologies.

Creating HTML Pages

In order to create a client-side application (an application that runs within a Web browser), you first need to learn how to create Web pages to house the application. This section describes basic HTML elements and their usage.

Basic HTML Elements

An HTML element, markup, or tag is a text identifier for some HTML object, such as an image, a link, or another HTML document.

Example: An empty HTML page (Basic.html).

```html
<HTML>
  <HEAD>
    <TITLE>Page Title Here</TITLE>
  </HEAD>
  <BODY>
    This is where the actual page lives
  </BODY>
</HTML>
```

As you can see in the Basic.html example, HTML elements have start and end tags that enclose the element content. The start tag consists of the element name enclosed in angle brackets. The end tag is the same as the start tag, except that it has a forward slash as a prefix to the element name. Some element types, however, do not require an end tag and some never have an end tag.

The following table describes some of the basic elements that are used in the examples in this guide.
Basic HTML Elements

<table>
<thead>
<tr>
<th>Object Type</th>
<th>Element Syntax</th>
<th>Description</th>
</tr>
</thead>
</table>
| HTML Document     | `<HTML>
</HTML>`   | Defines the HTML document.                                                  |
| HTML Header       | `<HEAD>
</HEAD>` | Includes information such as the document title, styles used in the document, and scripts. |
| Document Title    | `<TITLE>
</TITLE>` | Defines the page title that appears in the title bar of the Web browser.    |
| Document Body     | `<BODY>
</BODY>` | Contains the visible portion of the HTML page. All text, images, links, and other objects must appear within the `<BODY>` element. |
| Text              | `<BR>`          | Plain text does not require special element syntax.                         |
| Comments          | <!-- comment here--> | Comments are usually different syntax than standard HTML elements. Any text or HTML within a comment block does not appear on the document. |
| Line Break        | `<BR>`          | No closing element is required.                                             |
| Paragraph Break   | `<P>
</P>`      | Creates a larger break than a line break. The closing element tag is optional. |
| Horizontal Line   | `<HR>`          | No closing element tag is required.                                         |
| Text Headings     | `<H#>`          | Where # is replaced with a number from 1 to 6.                              |
| Bold Text         | `<B>`           | Makes text inside the element bold.                                         |
| Underline Text    | `<U>`           | Makes text inside the element underlined.                                   |

HTML Examples

The following examples illustrate some HTML concepts.

To try these and any of the other examples presented in this guide, open a text editor, such as Notepad, and type the text as shown. Then create a directory within the Operational Insight HTML directory (located by default in C:\Program_Files\Matrikon\ProcessSuite\ProcessNet\Web\HTML) called Samples, and save your documents there as example1.html and
example2.html. Additionally, all of the examples can be found in a zipped file on the installation CD.

You can view these HTML pages by double-clicking the file name. However, as the pages get more complex, you will need to view them using the Web server and a URL, rather than directly from the hard drive. To view the files through the Web server, type the following URL into the Web browser’s Address edit box, replacing <filename> with the name of the file:

http://localhost/ProcessNet/html/Samples/<filename>

Example: Simple page with a heading and multiple paragraphs (SimpleHeading.html).

```html
<HTML>
<HEAD>
<TITLE>Example 1</TITLE>
</HEAD>
<BODY>
<H1>Example 1</H1>
This is a sample HTML page.
<P>
It’s very simple.
</BODY>
</HTML>
```

Example: Simple page with underlined and bold text (SimpleUnderline.html).

```html
<HTML>
<HEAD>
<TITLE>Example 2</TITLE>
</HEAD>
<BODY>
<H1>Example 2</H1>
This is bold text</B>
BR <!-- This will look different than <P> -->
This is underlined</U>
</BODY>
</HTML>
```

Common HTML Elements

The previous section introduced some of the most basic HTML elements. This section describes slightly more complicated, but common, elements.

Many of these elements use attributes to define visual options, such as color. An attribute is a property associated with the element. The attributes and attribute values for an element appear before the final > of the start tag.

For example, the <BODY> element has an attribute that defines the background color for the Web page. To make a Web page red, the following syntax would be used in the body element:

```html
<BODY bgcolor = red>
```

The bgcolor attribute defines the color of the document.

The following table describes some common elements.
## Other Common HTML Elements

<table>
<thead>
<tr>
<th>Object Type</th>
<th>Element Syntax</th>
<th>Common Attributes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fonts</td>
<td><code>&lt;FONT&gt;</code></td>
<td>color face size</td>
<td>Changes the font of the text within the element.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(Deprecated. See “Dynamic HTML” on page 12 for alternatives to the FONT element.)</td>
</tr>
<tr>
<td>Table or Grid</td>
<td><code>&lt;TABLE&gt;</code></td>
<td>bgColor height</td>
<td>Tables are useful for aligning text and other objects, both vertically and horizontally.</td>
</tr>
<tr>
<td></td>
<td><code>&lt;TR&gt;</code> (Row)</td>
<td>width</td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>&lt;TD&gt;</code> (Cell)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>&lt;TH&gt;</code> (Header)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Image</td>
<td><code>&lt;IMG&gt;</code></td>
<td>src</td>
<td>Displays an image in the page.</td>
</tr>
<tr>
<td>Link (Anchor)</td>
<td><code>&lt;A&gt;</code></td>
<td>href</td>
<td>Inserts a link to another page, or another point in the current page.</td>
</tr>
</tbody>
</table>

**Example: Setting a font style** *(FontStyle.html)*.

```html
<HTML>
  <HEAD>
    <TITLE>Font Style Set Example</TITLE>
  </HEAD>
  <BODY>
    <FONT color=red face=Arial size=16>
      This is big red text.
    </FONT>
  </BODY>
</HTML>
```

**Example: Using a table to align text** *(Table.html)*.

```html
<HTML>
  <HEAD>
    <TITLE>Table Example</TITLE>
  </HEAD>
  <BODY>
    <TABLE border=1>
      <TR>
        <TD>This is text at 1,1 (X, Y)</TD>
        <TD>Here we are at 2,1</TD>
      </TR>
      <TR>
        <TD>On second row at 1,2</TD>
        <TD>Bottom right 2,2</TD>
      </TR>
    </TABLE>
  </BODY>
</HTML>
```
Example: Linking to another page (Link.html).

```html
<HTML>
  <HEAD>
    <TITLE>Link Example</TITLE>
  </HEAD>
  <BODY>
    Click <A href=example1.html>HERE</A> to open a different page.
  </BODY>
</HTML>
```

Advanced HTML Elements

The previous sections described some basic HTML elements and their usage. There are a few additional elements used when developing pages for Operational Insight.

### Advanced HTML Elements

<table>
<thead>
<tr>
<th>Object Type</th>
<th>Element Syntax</th>
<th>Common Attributes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Applet</td>
<td>&lt;APPLET&gt;</td>
<td>code, archive</td>
<td>Inserts a Java applet into the page, such as the Operational Insight trend.</td>
</tr>
<tr>
<td>Container (Div)</td>
<td>&lt;DIV&gt;</td>
<td></td>
<td>Acts as a container for other HTML objects.</td>
</tr>
<tr>
<td>Button</td>
<td>&lt;BUTTON&gt;</td>
<td>accessKey</td>
<td>Standard button.</td>
</tr>
</tbody>
</table>

Example: Inserting a PNJTrend applet from Operational Insight in the page (PNJTrend.html).

```html
<HTML>
  <HEAD>
    <TITLE>PNJTrend Example</TITLE>
  </HEAD>
  <BODY>
    <Applet id="PNJTrend" CODEBASE="/processnet/html/pnjtrend"
             code="PNJTrend.BasicTrend"
             archive="/ProcessNet/HTML/PNJTrend/PNJTrend.jar"
             width=10 height=10
             MAYSCRIPT
             >
      <PARAM name='ID' value='PNJTrend'>
      <PARAM name='HideCursorButton' value=1>
      <PARAM name='OnError' value='trendErrorHandler'>
      <PARAM name='OnFetchNewData' value='fetchNewDataHandler'>
      <PARAM name='OnGetEventData' value='fetchEventSourceDataHandler'>
      <PARAM name='OnMarkMouseIn' value='onMarkMouseIn'>
```

```html
</BODY>
</HTML>
```
Note
This example will only work if it is run through the Web server.

Dynamic HTML

Dynamic HTML, or DHTML, is HTML with an accessible object model applied to it. This means DHTML objects can be accessed using script, such as VBScript or JavaScript. DHTML and HTML are often used as interchangeable terms, and are used interchangeably in this guide.

Styles

Using DHTML, you can apply styles to objects to change their visual attributes and non-visual behaviors. All HTML and DHTML objects can have various style attributes applied to them, such as height, width, color, border, and position.

To add a style to an object, specify the style attribute in the opening element text. For example:

```html
<DIV style = "background:red; height:100px"> Hello </DIV>
```

The previous example displays a red box, 100 pixels (px) high, with the word “Hello” in it. The following table lists some common styles, but there are many more that are not listed.

<table>
<thead>
<tr>
<th>Style</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>background</td>
<td>Changes the background color. You can use either hexadecimal or descriptive color names.</td>
<td>background:red</td>
</tr>
<tr>
<td></td>
<td></td>
<td>background:#FF0000</td>
</tr>
<tr>
<td>font</td>
<td>Changes the font face, display characteristics, and size.</td>
<td>font:italic 10pt Arial</td>
</tr>
<tr>
<td>color</td>
<td>Text color.</td>
<td>color:green</td>
</tr>
</tbody>
</table>
To separate individual styles within the style attribute, use semi-colons.

Several of the previously listed styles, such as font or border, have many sub-styles within them. These sub-styles, such as font size, can be used independently of the main style. For example, to set the font size and face for an object, either of the following styles could be used:

```html
<DIV style = "font-size:20pt; font-family:Arial">
  Big
</DIV>
```

```html
<DIV style = "font:20pt Arial">
  Big
</DIV>
```

Similarly, you can apply a border style to the entire border, or just to one side of the border:

```html
<DIV style = "border-bottom:2px red solid">
  Top
</DIV>
```

```html
<DIV style = "border-top-color:red; border-top-style:solid">
  Top Color
</DIV>
```
For more information, refer to an HTML reference.

Using PNSP to Generate HTML

PNSP pages, also known as PNHTML, extend standard HTML pages, allowing you to create HTML pages containing data from back-end systems, such as relational databases, historians, and OPC servers.

PNSP can be used with any HTML document. To use PNSP, you must insert special PNSP elements into the HTML document. These elements are processed by the NetPortal (Web server) before being sent to the user’s browser. In this way, simple elements can be transformed into anything from simple values to large tables.

You must save all PNSP pages with the extension PNSP, and PNSP pages must be run through the Web server; for example: http://localhost/ProcessNet/html/samples/example7.pnsp. If you are not logged on, you will be prompted to log on before the PNSP page is displayed.

The following table describes the PNSP elements:

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[#DataSet]</td>
<td>Defines a SQL or tag-based query against an Operational Insight data source. A SQL query is used for relational data, and a tag-based query is used for historians and OPC servers.</td>
</tr>
<tr>
<td>[#Value]</td>
<td>Inserts a single value into the HTML page. This value can come from a predefined query, or from a tag in a back-end data source, such as OPC.</td>
</tr>
<tr>
<td>[#Band]</td>
<td>Creates a band of HTML for each row of the data returned from a data set.</td>
</tr>
<tr>
<td>[#Global]</td>
<td>Sets default values for all attributes used by the other elements.</td>
</tr>
</tbody>
</table>
Band Element

The band element creates a band of HTML for each row of data returned from a data set.

The band element has only two attributes:

**Band Element Attributes**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>datasrc</td>
<td>If the tag attribute is specified, this attribute is the name of a Operational Insight real-time tag data source. If the Field attribute is specified, this is the name of an existing data set definition.</td>
</tr>
<tr>
<td>Definition</td>
<td>The definition attribute contains the band’s definition. This can be any number of value tags, HTML tags, or simple text. You can also embed other bands within a band.</td>
</tr>
</tbody>
</table>

Example: Creating a table from the SQL query from the DataSet examples (Query1; PNSPBand.pnsp).

```html
<HTML>
  <HEAD>
    <TITLE>PNSP Table Example</TITLE>
    <DATASET_ID><DATASET_ID>Query1</DATASET_ID></DATASET_ID>
    <DATASET_DATASRC><DATASET_DATASRC>SQL DS</DATASET_DATASRC></DATASET_DATASRC>
    <DATASET_SQL><DATASET_SQL>select Company, State from customer</DATASET_SQL></DATASET_SQL>
  </HEAD>
  <BODY>
    <TABLE border=1>
      <TR>
        <TH>Company</TH>
        <TH>State</TH>
      </TR>
      <#Band datasrc=Query1 definition="<TR>
          <TD>[#Value Field=0]</TD>
          <TD>[#Value Field=1]</TD>
      </TR>"/>
    </TABLE>
  </BODY>
</HTML>
```
The placement of the Dataset element is not important, as long as it appears before the value or band elements that use it.

**Dataset Element**

The Dataset element defines a SQL or tag-based query against a Operational Insight data source. A SQL query is used for relational data, whereas a tag-based query is used for historians and OPC servers. The Dataset element has the following attributes.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>The query ID, referenced in Band and Value tags.</td>
</tr>
<tr>
<td>SQL</td>
<td>The SQL to use for a relational DB query. This attribute is not required when accessing a tag-based data source.</td>
</tr>
<tr>
<td>DataSrc</td>
<td>The Operational Insight data source against which the query is run.</td>
</tr>
<tr>
<td>TAGS</td>
<td>A comma-separated list of tag names for use with a tag-based query. If blank, the request is treated as a snapshot data request. This attribute is not required for SQL queries.</td>
</tr>
<tr>
<td>Start</td>
<td>The starting time for a historical query. This attribute is not required for SQL queries.</td>
</tr>
<tr>
<td>End</td>
<td>The ending time for a historical query. If blank, the current time is assumed. This attribute is not required for SQL queries.</td>
</tr>
<tr>
<td>Step</td>
<td>The time step parameter for a tag-based query. This value changes with the type of function requested. Refer to the Operational Insight data source drivers documentation for the appropriate values. This attribute is not required for SQL queries.</td>
</tr>
<tr>
<td>Function</td>
<td>The Operational Insight data source functions. Typical functions are interp, plot, and comp. This attribute is not required for SQL queries.</td>
</tr>
<tr>
<td>Command</td>
<td>Data access commands. The commands available are all those available through the NetPortal or NetDAC, including RPCs (Remote Procedure Calls), tag searches, and metadata queries.</td>
</tr>
<tr>
<td>Exec</td>
<td>Specifies the SQL to execute that does not return a result set.</td>
</tr>
</tbody>
</table>
Example: Relational data.
[#Dataset id=Query1 datasrc=SQL_DS
sql="select Company, State from customer"]

Example: Real-time values.
[#Dataset id=Query2 datasrc=OPC_Sim
tags="Random.Int1, Random.Int2"]

Example: History data.
[#Dataset id = Query3 datasrc=OPC_Sim
tags="Random.Int1, Random.Int2" start=-10M end=* step=1M function=interp]

If double quotes are required inside a query, they must be prefixed with an escape character: ". For example, assume that Query1 above requires quotes around the table name Customer:
[#Dataset id=Query1 datasrc=SQL_DS
sql="select Company, State from "customer"]

Example: Command attribute.
[#Dataset id=tagSearch datasrc=OPC_Sim
command = "rpc=TagSearch&maxRecords=500&tagnamemask=*"]

Example: Exec attribute.
[#Dataset id=insertFish datasrc=OPC_Sim
exec="insert into animals(NAME, SIZECOL, WEIGHT, AREA) values ('Gold Fish',2,1,'Fish Bowls')"]

Note
These examples define the query, but do not actually run the query.

ErrorText and Error Elements
Using the ErrorText and Error elements, you can specify how errors in the page are displayed.
These elements have the attributes shown in the following table.

ErrorText and Error Element Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>The HTML or text you want to place before the list of errors.</td>
</tr>
<tr>
<td>Definition</td>
<td>The definition attribute contains the definition of the bands for the errors. Each error message is placed in its own band, as defined by the definition attribute.</td>
</tr>
<tr>
<td>Footer</td>
<td>The HTML or text you want to place after the list of errors.</td>
</tr>
</tbody>
</table>

Multiple [#ErrorText] elements can appear in a page. If they do, all errors up to the point of the [#ErrorText] element are added to the document, and the
error list is cleared. When the next [#ErrorText] element is encountered, only errors generated since the last element are inserted into the document. All data access errors are reported as a group in the first [#ErrorText] element, regardless of where the data access element appears in the PNSP document. If no errors occur on the page, no error text is displayed.

In the following example, the Dataset element would generate an error owing to incorrect syntax: there is a spelling error in the SQL statement.

Example: Using [#ErrorText] and [#Error] (PNSPError.pnsp).

```
<HTML>
  <HEAD>
    <TITLE>PNSP Table Example</TITLE>
    [#Dataset id=Qry1 Datasrc=SQL_DS sql="select Company, State from customer"]
  </HEAD>
  <BODY>
    [#Value DataSrc=Qry1 Field=0]  
    [#ErrorText header="<H3>This page had errors</H3>"  
      definition="<BR>[#Error]<BR>"  
     ]
  </BODY>
</HTML>
```

**Global Element**

Using the global element, you can set default values for all the attributes used by the other elements, which can reduce the amount of duplicate information entered.

For example, instead of entering the same data source name many times in different value elements, the global element can be used to define it for all future value elements.

For example:

```
 [#Value datasrc = OPC_SIM tag = abc]  
 [#Value datasrc = OPC_SIM tag = def]  
 [#Value datasrc = OPC_SIM tag = ghi]  
```

Would become:

```
 [#Global datasrc = OPC_SIM]  
 [#Value tag = abc]  
 [#Value tag = def]  
 [#Value tag = ghi]  
```

The global element has all the attributes defined for the other elements.

**Import Element**

Imported files are merged into the document in place of the import element. This can simplify the addition of standard page headers, footers, and other features that must be maintained across multiple documents. The import element has the attribute shown in the following table.

Import Element Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>File</td>
<td>The file name and path of the file you want to import.</td>
</tr>
</tbody>
</table>
You must use a forward slash (/) as a path separator, not a backslash (\). A double
backslash separator is also acceptable (\\). If you use a single backslash as a path
separator, it is removed from the path, and a “file not found” error is generated.
If an invalid file name is entered, the rest of the PNSP is parsed properly, and an
error is generated.
Example: Importing common script and stylesheet files.

```html
<HTML>
  <HEAD>
    <!--Import common script and stylesheets-->
    [#Import file="CommonHTMLIncludes.txt"]
  </HEAD>
  <BODY>
    [#Import file="CorporateBanner.txt"]
    <BR>
    This is a sample report with standard corporate header and footer.
    <BR>
    [#Import file="LegalDisclaimer.txt"]
  </BODY>
</HTML>
```

**Login Element**

If you use an invalid user name or password, the login will fail, subsequent
operations will not be performed, and an error message will be generated. The
**login** element has the attributes shown in the following table.

**Login Element Attributes**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>User</td>
<td>The user name used to log on.</td>
</tr>
<tr>
<td>Password</td>
<td>The password used to log on (unencrypted).</td>
</tr>
</tbody>
</table>

Because the password is not encrypted, you are strongly advised to store pages
that use the **[#login]** element on the file system instead of the content
database. If these pages are stored in the database, users may be able to gain
access to the password using NetOffice or NetManage. Only one **login** element
should exist per page.
Although only the first **login** element in the page is used, all others are
evaluated.
Example: Using the **login** element.

```html
<HTML>
  <HEAD>
    <TITLE>
      Sample Report: All Users
    </TITLE>
  </HEAD>
  <BODY>
    [#Login
      User="ReportViewer"
      Password="please"]
    [#DataSet
      ID=ReportData
datasrc=SecureDB
```
Operational Insight R362.1 Programmer’s Guide

Permission Element

Using the permission element, you can enable, disable, or change functionality, based on the user’s permissions for the specified function. This element has the attributes shown in the following table.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key</td>
<td>The “key” name for the function (see “Modifying Access to Functions” in the Operational Insight Administrator’s Guide).</td>
</tr>
<tr>
<td>Function</td>
<td>The function name under the specified key.</td>
</tr>
<tr>
<td>Permit</td>
<td>The text you want to insert into the PNSP if the user has permission for the specified function.</td>
</tr>
<tr>
<td>Deny</td>
<td>The text you want to insert into the PNSP if the user does not have permission for the specified function.</td>
</tr>
</tbody>
</table>

If you use an invalid function or key name, PNSP inserts the “deny” text into the page and logs the error. Permissions are created and assigned to functions and keys using NetManage.

Example: Using the permission element.

```html
    <HTML>
    <BODY>
    Click OK to approve this request.
    <BR>
    [#Permission
    key="CustomApp"
    Function="ApproveRequest"
    permit="<BUTTON>OK</BUTTON>"
    deny=""
    ]
    <BUTTON>Cancel</BUTTON>
    </BODY>
    </HTML>
```

Example: Using password permissions.

```javascript
    function ShowPasswordPermissions()
    {
        [#Permission
        key="Generic"
        Function="ChangePassword"
        permit="alert('You may change your password');"
        deny="alert('You are not permitted to change your password');"
        ]
    }
```
Value Element

The value element inserts a single value into the HTML page. This value can come from a predefined query, or from a tag in a back-end data source, such as OPC.

The value element has the attributes shown in the following table.

Value Element Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tag</td>
<td>The name of the tag for which you want to get a real-time value. This is not required if the value field is referencing an existing data set.</td>
</tr>
<tr>
<td>Field</td>
<td>If you are referencing an existing data set definition, this is either a field name in the data set’s result set, or a zero-based field number. This attribute is not required if you are referencing a real-time tag source.</td>
</tr>
<tr>
<td>DataSrc</td>
<td>If the Tag attribute is specified, this attribute is the name of a Operational Insight real-time tag data source. If the Field attribute is specified, this is the name of an existing data set definition, and is optional if the value tag appears in a band definition.</td>
</tr>
<tr>
<td>Attr</td>
<td>The Attr can be specified as Value (default), which returns the value of the tag; Status which returns the numeric status of the tag; Statstring, which returns the status as a string; or Timestamp, which returns the time stamp of the tag.</td>
</tr>
<tr>
<td>Format</td>
<td>An optional format specifier for numerical data. The default format for numerical data is width.precision. For dates and times, you can use the standard m, d, y and h, n, s characters to build a date/time display.</td>
</tr>
</tbody>
</table>

Example: Value elements for tag-based data sources (PNSPValue.pnsp).

```html
<HTML>
  <HEAD>
    <TITLE>PNSP Value Example</TITLE>
  </HEAD>
  <BODY>
  </BODY>
</HTML>
```

The previous example assumes that there is a Operational Insight data source called OPC_Sim defined, and that it has a tag within it called Random.Int2. The output of the example would look like:

Value for Random.Int2: 1234
Use the following elements to retrieve and display values in the browser window from one of the data sets defined in the data set examples:

[#Value DataSrc = Query1 Field = 0]

or

[#Value DataSrc = Query1 Field = "State"]

**Variable Element**

Using the variable element, you can insert the contents of a Operational Insight global variable. The variable element has the attributes shown in the following table.

### Variable Element Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Category</td>
<td>The variable category, as defined in NetManage.</td>
</tr>
<tr>
<td>Name</td>
<td>The name of the variable in the category.</td>
</tr>
<tr>
<td>Encoding</td>
<td>The encoding of the global variable content string for use in different insertion contexts, with appropriate escaping and characters. Available values are: HTML, JSSString, JavaScript, and HTTPEncode. The default value is HTML.</td>
</tr>
</tbody>
</table>

If you enter an invalid variable or category name, no text is inserted for the [#Variable] element, and an error is generated.

**Example: Using the Variable element.**

```html
<HTML>
<BODY style="background-image:URL('[#Variable category="Corporate styles" name="BackgroundImage"]');">
<br>
<H3>
Welcome to Operational Insight!
</H3>
<br>
<a href="mailto: [#Variable category="General" name="AdminEmail"]?subject="Help me out here!">Help</a>
</BODY>
</HTML>
```

**Using Parameters with PNSP**

When calling a page that uses PNSP elements, you can use parameters in the URL to pass attribute values to the PNSP elements on the page.

For example, to set up a page to receive a tag name through the URL, the value element would look like:

[#value tag = *[tag]]

When referencing the page through a URL, you would specify the tag name:
Scripting

You can use client-side scripting to get the most out of DHTML and Operational Insight. By using scripts that run within the Web browser, you can provide users with an interactive environment that does not rely on reloading Web pages to update displays based on user input.

The combination of script and DHTML is powerful. For example, Operational Insight’s NetDraw tool is built entirely using JavaScript and DHTML.

VBScript vs. JavaScript

VBScript and JavaScript are the two dominant scripting languages available in the Web browser. Both have strengths and weaknesses, and the choice of which to use is usually a personal preference. Both script languages can be used interchangeably: JavaScript can call VBScript functions, and vice versa.

Two of the major differences between the two languages are syntax and error handling. JavaScript, as the name implies, is similar syntactically to Java and C/C++. VBScript, as its name also implies, is syntactically similar to VB and VBA. VBScript uses the same error handling constructs as VB (on..error), which is generally considered a poorer mechanism than the try..catch mechanism that JavaScript uses.

This guide uses mainly JavaScript for examples, with some VBScript shown for comparative purposes.

Creating Scripts in HTML Pages

You can use the script element to add scripts to HTML pages. For example:

```html
<SCRIPT language='VBScript'>
Sub Test
    MsgBox 'Hello'
End Sub
</SCRIPT>
```

The language attribute is optional if you use JavaScript. The script element is normally added to the HTML header block between the `<HEAD>` and `</HEAD>` elements.

The script element can also be used to reference scripts stored in files separate from the HTML file. You can create reusable scripts and then reference them from any HTML page.

For example, to reference the scripts contained in a file called `my_scripts.js`, the syntax would be:

```html
<SCRIPT src='my_scripts.js'>
</SCRIPT>
```

You can also insert scripts inline with HTML in the document. In this case, the script runs as the page is loading, so the script can dynamically generate or append to the document. This can be useful when multiple elements of the same type need to be inserted.
Example: Inline scripts (Script.html).

```html
<HTML>
  <BODY>
    <SCRIPT language=VBScript>
      document.write "Hello world!"
    </SCRIPT>
  </BODY>
</HTML>
```

The `document.write` method inserts the text “Hello World” into the HTML page at the point where the script element appears.

**Accessing HTML Objects from Script**

In order for a script to gain access to an HTML object directly, the object must have an identifier associated with it. This identifier is specified using the `id` attribute.

For example:

```html
<DIV id = theDiv>
  This is the div
</DIV>
```

Using the `id`, scripts can gain access to all the properties and methods for the HTML object.

For example, the following JavaScript would display the text contained within the div in the previous example:

```javascript
alert( theDiv.innerText );
```

The corresponding VBScript would be:

```vbscript
MsgBox theDiv.innerText
```

The `innerText` property is one of many properties available in DHTML. Although the remaining examples in this guide illustrate the usage of many of these properties, it is beyond the scope of this guide to cover all the methods and properties for HTML objects.

---

**FMI**

For more information, refer to a DHTML reference.

---

**Handling HTML Events**

All HTML objects have specific events associated with them. Different objects have different events, but the methods for referencing these events is the same.

JavaScript provides two common methods for handling events: defining an event handler based on the object name, or executing inline script. VBScript enables you to create an event handler only, but since JavaScript can call VBScript procedures, functions can also be executed in-line.

Example: Creating an event handler in JavaScript (EventJS.html).

```html
<HTML>
  <HEAD>
    <SCRIPT defer=true>
      function btnMyButton.onclick()
      {
```

---
As the previous example shows, to create an event handler for an object, you must:

- Defer the script. This prevents the script from being examined until the entire HTML page has loaded.
- Define an ID for the HTML object. In this case, the ID is `btnMyButton`.
- Create the event handler using the syntax `id.Event Name`.

Example: Creating an event handler in VBScript (`EventVB.html`).

```html
<HTML>
<HEAD>
<SCRIPT defer=true language=VBScript>
Sub btnMyButton_onclick()
    MsgBox "Clicked"
End Sub
</SCRIPT>
</HEAD>
<BODY>
    <BUTTON id=btnMyButton>Click Me</BUTTON>
</BODY>
</HTML>
```

The syntax for naming the event handler in JavaScript and VBScript is similar.

Example: Running in-line script (`InlineScript.html`).

```html
<HTML>
<HEAD>
</HEAD>
<BODY>
    <BUTTON onclick="alert('Clicked')">Click Me</BUTTON>
</BODY>
</HTML>
```

In this example, no custom function is called. The message box is displayed directly within the object's definition. JavaScript uses single and double quotes interchangeably, which is useful when using quotes within text that is already quoted.

Example: Taking action when the page loads (`OnLoad.html`).

```html
<HTML>
<HEAD>
</HEAD>
<BODY onload="doOnLoaded()">
    <BUTTON onclick="doOnLoaded()">Click Me</BUTTON>
</BODY>
</HTML>
```

Example: Accessing object properties in an event (`ObjectProperties.html`).
<HTML>
  <HEAD>
    <SCRIPT defer=true>
      function btnMyButton.onclick()
      {
        btnMyButton.innerText = "Clicked";
      }
    </SCRIPT>
  </HEAD>
  <BODY>
    <BUTTON id=btnMyButton>Click Me</BUTTON>
  </BODY>
</HTML>
Chapter 3: NetPortal Data Access Interface

The NetPortal provides several methods you can use to retrieve data from Operational Insight after the page has loaded, including URL queries, HTML forms, and script.

Note
Some of the examples in this chapter make use of the MatrikonOPC Server for Simulation.

URL Queries

URL Query refers to typing a URL that includes commands for data access into the address box of the browser. The result of the query can be formatted in a number of ways, including HTML, Excel, CSV, XML, and binary.

URL Query Syntax

Almost all queries to Operational Insight use the same syntax, which includes the path to the server, the data source, the command (or query), and the format for the returned data:

http://(server name)/(netportal)/data/(datasource)?(command)&(format)

In this example, (netportal) is the path to the NetPortal. By default, this is:

/ProcessNet/isapi/netportal/netportal.dll

The commands used to retrieve data are discussed in detail in the following sections.

Format Specifiers

Operational Insight provides several data presentation formats that can be used with all data requests. The following table describes these formats.
Chapter 3: NetPortal Data Access Interface

Formats

<table>
<thead>
<tr>
<th>Format Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTML</td>
<td>Displays the returned data in an HTML table.</td>
</tr>
<tr>
<td>Excel</td>
<td>Launches Excel and displays the data in a spreadsheet.</td>
</tr>
<tr>
<td>CSV</td>
<td>Displays the data as comma-separated values.</td>
</tr>
<tr>
<td>ADO</td>
<td>Displays the data as XML, using the Rowset schema compatible with Microsoft ADO.</td>
</tr>
<tr>
<td>NST or no format</td>
<td>Default. Binary format used by the WebDataSet. Not human-readable.</td>
</tr>
</tbody>
</table>

FMI
For more information about creating custom formats, see “Creating Packet Providers” on page 159.

Data Access Commands

All requests to retrieve data from the NetPortal are sent to the data path (or entry point) of the NetPortal:

http://.../netportal.dll/data/

The data path is followed by the data source name, a question mark, and the list of commands and parameters for the command. The two most basic request types for data are SQL-based and tag-based.

SQL-based Data Access

To gain access to relational databases, such as Oracle, SQL Server, or Access, you must use SQL. You also require a SQL reference for the database to which you are connecting, because all SQL statements are passed through Operational Insight to the database. Operational Insight does not process any of the SQL statements.

SQL Command Syntax

To access SQL, you must specify the SQL to be executed, using either the SQL keyword or the EXEC keyword. Use the SQL keyword when a result set (data) will be returned from the database, such as with a SELECT statement. Use EXEC when the database will not return a result set, such as with the UPDATE or DELETE statements.

In addition to the SQL statement, you must also specify the format for the returned data (see “Format Specifiers” on page 27). Otherwise, a binary result set will be returned, which is not viewable in the Web browser.
To gain access to the Interbase database installed with Operational Insight, using a URL, type:

```
http://[svr]/[path]/netportal.dll/data/SQL_DS?sql=select*fromindustry&format=html
```

Where [svr] is the name of the Operational Insight computer and [path] is the server’s path to NetPortal, which by default is:

```
ProcessNet/isapi/netportal/
```

In the previous example, the format specifier used is HTML. This causes Operational Insight to format the results of the query in an HTML table. Different formats can be specified to view the data in other formats.

For example, to see the data in Excel instead of an HTML table, modify the command to:

```
http://[svr]/[path]/netportal.dll/data/SQL_DS?sql=select*fromcustomer&format=Excel
```

The following example adds a record to the industry table. In this example, EXEC is used instead of SQL, causing the data returned to indicate the success or failure of the operation only:

```
http://[svr]/[path]/netportal.dll/data/SQL_DS?Exec=insert into industry values(0, 'Chem', 'Chemicals')&format=html
```

### Writing BLOB Data

**Note**

Operational Insight also provides programmatic access to data from server-side scripts and applications, as described in “Writing BLOB Data Using NetDAC” on page 152.

BLOB (Binary Large OBject) data occurs in many databases. BLOBs are used to store large amounts of text, images, and other data that could be of variable size. Because normal SQL statements cannot write to BLOB fields directly, a separate method is used. The command to write BLOB data consists of the parameters shown in the following table.

#### BLOB Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Sample Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>sql</td>
<td>Specifies the SQL statement to be executed to write the BLOB. A question mark (?) is used as a placeholder for the BLOB data.</td>
<td>exec = update someTable set theBlob =?</td>
</tr>
<tr>
<td>format</td>
<td>Specifies the presentation format.</td>
<td>format = HTML</td>
</tr>
<tr>
<td>blobData</td>
<td>The BLOB data to write to the database.</td>
<td></td>
</tr>
</tbody>
</table>
The result set returned after writing a BLOB field consists of a single record with two columns: **Success** and **Message**. The **Success** field contains `true` if the operation succeeded, and `false` if it failed. The **Message** field contains the applicable error message if the operation failed.

**Example:** Writing a BLOB field to the **Venues** table.

```
http://[svr]/[path]/netportal.dll/data/SQL_DS?sql=update venues set Remarks=? where VenueNo=4&blobData=Some data&format=html&rpc=writeBLOB
```

**Tag-based Data Access**

There are three categories of tag-based data access:

- Real-time (snapshot) queries
- Historical data queries
- Advanced historical queries
Note
The time stamp on the data is displayed based on the client’s time zone. If
you are using Integrated Login, you have to login to the OI landing page
(http://<server name>/processnet) and run the query, to display the data in
the client’s timezone.

Real-time Data Queries
Real-time queries, also called snapshot data, retrieve the last value written to the
data source and use the parameters shown in the following table.

Real-Time Data Query Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Sample Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>tag#</td>
<td>The tag for which you want to get data, where # starts at 0 and increases for each tag requested.</td>
<td>tag0 = MyTag &amp;tag1 = AnotherTag</td>
</tr>
<tr>
<td>format</td>
<td>The presentation format.</td>
<td>format = HTML</td>
</tr>
<tr>
<td>SnapTime</td>
<td>If specified, returns snapshot time stamps for the tags. If this is not specified, only the time stamp that represents the request time is returned.</td>
<td>SnapTime = true</td>
</tr>
<tr>
<td>Refresh</td>
<td>If specified, sets the update/refresh rate (in ms) on the OPC group created for a real-time data query against an OPC server. Optimally, should be set to twice as fast (that is, half the value) as the actual interval between queries (the interval used for client polling of the Operational Insight server). If this is not specified, the default refresh rate is 1,000 ms (1 sec).</td>
<td>Refresh = 5000</td>
</tr>
</tbody>
</table>

Example: Reading real-time values from the OPC simulation server installed with Operational Insight.

http://[svr]/[path]/netportal.dll/data/OPC_Sim?tag0=Random.Int2&tag1=Random.Int1&format=html
http://[svr]/[path]/netportal.dll/data/OPC_Sim?tag0=Random.Int2&tag1=Random.Int1&SnapTime=true&format=html&refresh=5000
The result set returned from a real-time data request consists of a single row with columns representing the tag value, status, and time stamp.

For example, a request for two tags, “TagA” and “TagB”, with the SnapTime parameter specified, would result in the data shown in the following table being returned to the Web browser:

<table>
<thead>
<tr>
<th>TIMESTAMP</th>
<th>TagA. VALUE</th>
<th>TagA. STATUS</th>
<th>TagA. TIMESTAMP</th>
<th>TagB. VALUE _1</th>
<th>TagB. STATUS _1</th>
<th>TagB. TIMESTAMP _1</th>
</tr>
</thead>
<tbody>
<tr>
<td>4/21/02</td>
<td>123.4</td>
<td>192</td>
<td>4/21/02 11:29:33</td>
<td>453.21</td>
<td>192</td>
<td>4/21/02 11:29:13</td>
</tr>
</tbody>
</table>

**Historical Data Queries**

Historical data queries allow you to retrieve a range of data from historians and other tag-based data sources. Historical queries require the parameters shown in the following table.

**Historical Data Query Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Sample Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>tag#</td>
<td>The tag for which you want to get data, where # starts at 0 and increases for each requested tag. Some functions do not support retrieval of more than one tag per query.</td>
<td>tag0 = MyTag &amp;tag1 = AnotherTag</td>
</tr>
<tr>
<td>format</td>
<td>The presentation format.</td>
<td>format = HTML</td>
</tr>
<tr>
<td>start</td>
<td>The starting date and time for the data. This can be in a relative time format or an explicit format.</td>
<td>start = *-1D</td>
</tr>
<tr>
<td>end</td>
<td>The end date and time for the data.</td>
<td>end = *</td>
</tr>
<tr>
<td>function</td>
<td>The method or aggregate you want to use when requesting data. Different data sources support different functions, but most support raw and interp.</td>
<td>function = interp</td>
</tr>
</tbody>
</table>
Data Access Commands

For more information about data source configuration, see “Administering Data Sources” in the Operational Insight Administrator’s Guide.

Example: Reading interpolated (sampled) history values from the OPC simulation server installed with Operational Insight. The OPC simulation server does not store history and, therefore, returns the same value for all periods in history.

http://[svr]/[path]/netportal.dll/data/OPC_Sim?tag0=Random.Int2&start=*&end=*&function=interp&step=1H&format=html

Example: Reading raw history values from the OPC simulation server.

http://[svr]/[path]/netportal.dll/data/OPC_Sim?tag0=Random.Int2&start=*&end=*&function=comp&format=html

Advanced Data Access Commands

The previous data history queries are simple in that they retrieve the values in ascending time order (oldest to newest value). Operational Insight also provides several advanced data access functions for history data that can do more than simply retrieve sequential data.

The following table describes these additional query types.

<table>
<thead>
<tr>
<th>Query Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rawTagHistory</td>
<td>Retrieves the raw data for a tag.</td>
</tr>
<tr>
<td>processedTagHistory</td>
<td>Retrieves sampled data for one or more tags.</td>
</tr>
</tbody>
</table>

Although the advanced methods provide greater control over how the data is accessed, the basic functionality of rawTagHistory and processedTagHistory can be accomplished with simple history queries. For
example, the start and end times can be reversed, resulting in data being returned in descending order (newest value first to oldest value).

**Using rawTagHistory**

Use the parameters shown in the following table with the `rawTagHistory` request.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Sample Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>tag or tag0</td>
<td>The tag for which you want to retrieve data.</td>
<td><code>tag0 = Random.Int2</code></td>
</tr>
<tr>
<td>start</td>
<td>The start time for the data.</td>
<td><code>start = *-1d</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>start = 2002/04/21 11:33</code></td>
</tr>
<tr>
<td>end</td>
<td>The end time for the data.</td>
<td><code>end = *</code></td>
</tr>
<tr>
<td>timeBounds</td>
<td>If true, yes, or 1, specifies that data bounding the specified time should be included. This is specific to OPC data sources.</td>
<td><code>timeBounds=true</code></td>
</tr>
<tr>
<td>maxRecords</td>
<td>The maximum number of values you want returned. If this is not specified, all values are returned up to the maximum number of values defined in the data source configuration.</td>
<td><code>maxRecords = 1000</code></td>
</tr>
<tr>
<td>rpc</td>
<td>Used to specify the query you want to make. In this case, it would be set to <code>rawTagHistory</code>.</td>
<td><code>rpc = rawTagHistory</code></td>
</tr>
<tr>
<td>timeZone</td>
<td>The offset of the client from GMT, in minutes. This is filled in automatically by Operational Insight, but can be over-ridden. It can be used in any query dealing with tag data, and has no effect on SQL data queries.</td>
<td></td>
</tr>
</tbody>
</table>

**FMI**

For more information about data source configuration, see “Administering Data Sources” in the *Operational Insight Administrator’s Guide*.
Example: Reading raw history values using `rawTagHistory`.
```
http://[svr]/[path]/netportal.dll/data/OPC_Sim?tag0=Random.Int2&start=*-1d&end=*&rpc=rawTagHistory&format=html
```
Example: Reading the first value in the archive written after April 1 at 1PM.
```
http://[svr]/[path]/netportal.dll/data/OPC_Sim?tag0=Random.Int2&start=2002/4/1 13:00&maxRecords=1&rpc=rawTagHistory&format=html
```
Example: Reading the last value written to the archive (after 1970).
```
http://[svr]/[path]/netportal.dll/data/OPC_Sim?tag0=Random.Int2&start=*&end=1970/01/01&maxRecords=1&rpc=rawTagHistory&format=html
```

Using `processedTagHistory`

Use the parameters shown in the following table with `processedTagHistory`.

**Processed Tag History Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Sample Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>tag#</code></td>
<td>The tags for which you want to retrieve data, where # starts at 0, and increases for each tag.</td>
<td><code>tag0 = Random.Int2</code> &amp; <code>tag1 = Random.Int1</code></td>
</tr>
<tr>
<td><code>start</code></td>
<td>The start time for the data.</td>
<td><code>start = *-1d</code> or <code>start = 2002/04/21 11:33</code></td>
</tr>
<tr>
<td><code>end</code></td>
<td>The end time for the data.</td>
<td><code>end = *</code></td>
</tr>
<tr>
<td><code>maxRecords</code></td>
<td>The maximum number of values you want to return. If this is not specified, all values are returned up to the maximum number of values defined in the data source configuration.</td>
<td><code>maxRecords = 1000</code></td>
</tr>
<tr>
<td><code>aggregate</code> or <code>function</code></td>
<td>The aggregate (or function) used to retrieve data from the data source. Functions vary by data source, but common functions are <code>interp</code>, <code>min</code>, <code>max</code>, <code>std_dev</code>.</td>
<td><code>aggregate = interp</code></td>
</tr>
</tbody>
</table>
Chapter 3: NetPortal Data Access Interface

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Sample Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>sampleInterval</td>
<td>The interval between samples. This can be in seconds, minutes, hours, days, or years.</td>
<td>sampleInterval = 30M</td>
</tr>
<tr>
<td>or step</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rpc</td>
<td>Used to specify the type of query you want to make. In this case, it would be set to processedTagHistory.</td>
<td>rpc = processedTagHistory</td>
</tr>
<tr>
<td>timeZone</td>
<td>The offset of the client from GMT, in minutes. This is filled in automatically by Operational Insight, but can be over-ridden. It can be used in any query dealing with tag data, and has no effect on SQL data queries.</td>
<td></td>
</tr>
</tbody>
</table>

FMI
For more information about data source configuration, see “Administering Data Sources” in the Operational Insight Administrator’s Guide.

Example: Reading interpolated values using processedTagHistory.
http://[svr]/[path]/netportal.dll/data/OPC_Sim?tag0=Random.Int2&start=*&-1d&end=*&sampleInterval=10M&aggregate=interp&rpc=processedTagHistory&format=html

Example: Reading interpolated values in reverse time order.
http://[svr]/[path]/netportal.dll/data/OPC_Sim?tag0=Random.Int2&start=*&end=*-1d&sampleInterval=10M&aggregate=interp&rpc=processedTagHistory&format=html

Abstracting Data Sources
Operational Insight has the ability to access tag-based data sources using SQL, and to access relational data sources using tag syntax.

Using SQL to Retrieve Tag Data
Tag-based data sources (OPC, historians) can be treated as relational databases with the tables and fields described in the following tables.
History Table: Contains Historical Data

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>timestamp</td>
<td>The data time stamp. Can be used in <code>select</code> and <code>where</code> clauses.</td>
</tr>
<tr>
<td>value</td>
<td>The data value for each time stamp.</td>
</tr>
<tr>
<td>status</td>
<td>The data status for each time stamp.</td>
</tr>
<tr>
<td>tag</td>
<td>The tag for the request. Used only in the <code>where</code> clause.</td>
</tr>
<tr>
<td>function</td>
<td>The aggregate to be used when requesting data.</td>
</tr>
<tr>
<td>step</td>
<td>The time interval for the chosen aggregate.</td>
</tr>
</tbody>
</table>

Snapshot Table: Contains Real-time Data

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>timestamp</td>
<td>The data time stamp. Can be used in the <code>select</code> and <code>where</code> clauses.</td>
</tr>
<tr>
<td>value</td>
<td>The data value for each time stamp.</td>
</tr>
<tr>
<td>status</td>
<td>The data status for each time stamp.</td>
</tr>
<tr>
<td>tag</td>
<td>The tag for the request. Used only in the <code>where</code> clause.</td>
</tr>
</tbody>
</table>

Example: Reading interpolated values from OPC using SQL.

http://[svr]/[path]/netportal.dll/data/OPC_Sim?sql=select*from history where tag=Random.Int2 and timestamp>"*-1d"and function=interp and step=0M&format=html

Example: Reading real-time values from OPC using SQL.

http://[svr]/[path]/netportal.dll/data/OPC_Sim?sql=select*from snapshot where tag=Random.Int2&format=html

Using Tag-Based Commands to Get Relational Data

The standard tag-based data access commands will work with SQL data sources, as described in "Tag-based Data Access" on page 30.

Metadata Access Commands

Metadata is the data that describes the elements of a data source. For example, relational databases provide information about tables, fields and field data types, primary keys, and more. Tag-based data sources, such as OPC, provide tag lists and tag attribute information.

Much like data access commands, there are two types of metadata requests: basic and advanced. The basic commands are less specific and are easier to use, but the advanced commands provide more control over the information retrieved.
Basic Metadata Requests

You can use basic metadata requests to retrieve data source names, tag lists, and table names.

Retrieving Data Source Names

You can retrieve a list of data source names from the NetPortal by using the special data source name `DataSources` in the data path.

Use the parameters shown in the following table.

Parameters for Retrieving Data Source Names

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Sample Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>tag</td>
<td>If true, returns all tag-based data sources, and all relational data sources configured to look like tag data sources.</td>
<td>tag = true</td>
</tr>
<tr>
<td>sql</td>
<td>If true, returns all SQL-based data sources.</td>
<td>sql = true</td>
</tr>
<tr>
<td>infoType</td>
<td>An integer representing either Operational Insight driver metadata, or an ODBC metadata request. For a list of these integers and their corresponding description strings, see “Metadata for getInfo Request” on page 42.</td>
<td>infoType = -112</td>
</tr>
</tbody>
</table>

Example: Retrieving a list of SQL-based data sources.

```
http://[svr]/[path]/netportal.dll/data/DataSources?sql=true&format=html
```

Example: Retrieving a list of all data sources.

```
http://[svr]/[path]/netportal.dll/data/DataSources?sql=true&tag=true&format=html
```

The result set returned from the query contains three columns as shown in the following table.

Result Set When Retrieving Data Source Names

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title</td>
<td>Name of the data source as configured in Operational Insight.</td>
</tr>
<tr>
<td>Type</td>
<td>The type of data source. Valid types are Tag, SQL, and TagToSQL.</td>
</tr>
</tbody>
</table>
Retrieving a List of Tags

To retrieve a list of tags from a tag-based data source, a request resembling a standard data access request is used, with the function set to `TagSearch`. The parameters shown in the following table are used.

### Parameters for Retrieving a List of Tags

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Sample Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>tag0 or tagNameMask</td>
<td>The search mask for the tag name, which may include wildcard characters * and ?.</td>
<td>tag0 = s*</td>
</tr>
<tr>
<td>tag1 or desc</td>
<td>A search mask for the descriptor of the tag.</td>
<td>tag1 = <em>my tag</em></td>
</tr>
<tr>
<td>function or rpc</td>
<td>Identifies the tag list request. Must be set to <code>TagSearch</code>. If <code>rpc</code> is used rather than function, <code>maxRecords</code> is required.</td>
<td>function = TagSearch</td>
</tr>
<tr>
<td>maxRecords</td>
<td>The maximum number of tags to return in the search. Default is 1,000. Optional, unless <code>rpc</code> parameter is used.</td>
<td>maxRecords = 500</td>
</tr>
<tr>
<td>intfType</td>
<td>(Optional) Applies to OPC data sources. It is the interface type to be used for doing tag searches, either real-time (0) or historical (1). Defaults to real-time.</td>
<td>intfType = 1</td>
</tr>
</tbody>
</table>

Example: Retrieving a list of tags from the OPC simulation server.

http://[svr]/[path]/netportal.dll/data/OPC_SIM?function=TagSearch&tag0=*&tag1=*&format=html

Example: Retrieving a list of tags from the OPC simulation server with a specific description.

http://[svr]/[path]/netportal.dll/data/OPC_SIM?tag0=*&tag1= *random*&function=TagSearch&format=html

The result set returned for the tag list contains three columns: the tag ID, the tag name, and the tag description. If the specified data source is of type SQL, the result set will instead be one column, `TableName`, which will list the available tables for that data source.
Retrieving a Hierarchical List of Tags

To retrieve a hierarchical list of tags from a tag-based data source, the function is set to TagSearch, the same way as when retrieving a flat list of tags. The parameters shown in the following table are used.

## Parameters for Retrieving a Hierarchical List of Tags

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Sample Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>doHierarchal</td>
<td>Returns a record set that can be used to display the tags in a tree. This record set can be passed directly to the tree applet in a normalize or normalize_append call. Only applies to data sources of type Tag, which support infoType=-112 (INFO_TAG_BROWSE_HIERARCHAL). Optional, with values of either 1 (true) or 0 (false).</td>
<td>doHierarchal = 1</td>
</tr>
<tr>
<td>folderPath</td>
<td>The path to the folder that will be the starting point for tree expansion when using doHierarchal.</td>
<td>folderPath = path to starting point folder</td>
</tr>
</tbody>
</table>

**FMI**

For more information about the infoType parameter, see “Metadata for getInfo Request” on page 42.

---

Example: Retrieving a hierarchical list of tags from the OPC simulation server.

http://[svr]/[path]/netportal.dll/data/OPC_SIM?tag0=*&function=TagSearch&doHierarchal=true

If there are too many tags to be returned, the doHierarchal parameter will only return one level of folders. In order to expand the tree further, additional queries might be required. The folderPath parameter is used in this case, to continue tree expansion. Each time a tree node (folder) is expanded, check for the dummy node Loading. If it exists, there is another level of expansion that can be done, as follows:

1. Delete the dummy node Loading.
2. Perform another doHierarchal query, specifying as the folderPath the path to the folder in which the dummy node appeared.
3. Use the normalize_append applet to add the next level (or the rest of the levels, if there were not too many for this query) of items to the tree.
4. Repeat this procedure for each dummy node, expanding the tree until there are no more dummy nodes.
Example: Retrieving a hierarchical list of tags from the OPC simulation server with a specific folderPath as the starting point.

http://[svr]/[path]/netportal.dll/data/
OPC_SIM?tag0=*&function=TagSearch&doHierarchal=true&folderPath=<path to dummy node parent folder>

Retrieving a List of Functions

Using the NetPortal, you can retrieve a list of functions (aggregates) for tag-based data sources. This request uses only a single parameter: function. The function parameter is set to TableList to retrieve the available functions.

Example: Retrieving the list of aggregate functions supported by OPC.

http://[svr]/[path]/netportal.dll/data/
OPC_SIM?function=TableList&format=html

Retrieving a List of Tables

Retrieving a list of tables for a data source requires the same syntax as retrieving a list of functions for a tag-based data source. The result set returned consists of one column, TableName, which holds comma-delimited values in the format function, description.

Example: Retrieving a list of tables in the sample Interbase database.

http://[svr]/[path]/netportal.dll/data/
SQL_DS?function=TableList&format=html

Advanced Relational Database Metadata

Use the functions shown in the following table to obtain advanced metadata from ODBC data sources.

### Functions for Obtaining Advanced Metadata

<table>
<thead>
<tr>
<th>Query Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getInfo</td>
<td>Returns general information about the driver and data source.</td>
</tr>
<tr>
<td>tables</td>
<td>Returns a list of tables for the data source.</td>
</tr>
<tr>
<td>columns</td>
<td>Returns a list of fields for a specified table.</td>
</tr>
<tr>
<td>columnPrivileges</td>
<td>Returns a list of columns and associated privileges for a specified table.</td>
</tr>
<tr>
<td>foreignKeys</td>
<td>Returns a list of foreign keys for a specified table.</td>
</tr>
</tbody>
</table>
To identify each of these requests, you must use an `rpc` parameter. For example:

```
http://[svr]/[path]/netportal.dll/data/SQL_DS?rpc=tables&format=html
```

### getInfo Function

The `getInfo` function accepts a single parameter, `infoType`. The `infoType` parameter is an integer and represents either Operational Insight driver metadata, or an ODBC metadata request, as described in the following table.

#### Metadata for `getInfo` Request

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-112</td>
<td>INFO_TAG_BROWSE_HIERARCHAL</td>
</tr>
<tr>
<td>-111</td>
<td>INFO_TRANSACTION</td>
</tr>
<tr>
<td>-110</td>
<td>INFO_TAG_HDA_BROWSE</td>
</tr>
<tr>
<td>-109</td>
<td>INFO_TAG_HDA_WRITE</td>
</tr>
<tr>
<td>-108</td>
<td>INFO_TAG_HDA_READ</td>
</tr>
<tr>
<td>-107</td>
<td>INFO_TAG_DA_BROWSE</td>
</tr>
<tr>
<td>-106</td>
<td>INFO_TAG_DA_WRITE</td>
</tr>
<tr>
<td>-105</td>
<td>INFO_TAG_DA_READ</td>
</tr>
<tr>
<td>-104</td>
<td>INFO_BATCH_PARAMETERS</td>
</tr>
<tr>
<td>-103</td>
<td>INFO_LIVE</td>
</tr>
<tr>
<td>-102</td>
<td>INFO_BLOB</td>
</tr>
<tr>
<td>-101</td>
<td>INFO_SQL</td>
</tr>
<tr>
<td>0</td>
<td>SQL_MAX_DRIVER_CONNECTIONS</td>
</tr>
<tr>
<td>1</td>
<td>SQL_ACTIVE_STATEMENTS</td>
</tr>
<tr>
<td>2</td>
<td>SQL_DATA_SOURCE_NAME</td>
</tr>
<tr>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>-------</td>
<td>-------------</td>
</tr>
<tr>
<td>3</td>
<td>SQL_DRIVER_HDBC</td>
</tr>
<tr>
<td>4</td>
<td>SQL_DRIVER_HENV</td>
</tr>
<tr>
<td>5</td>
<td>SQL_DRIVER_HSTMT</td>
</tr>
<tr>
<td>6</td>
<td>SQL_DRIVER_NAME</td>
</tr>
<tr>
<td>7</td>
<td>SQL_DRIVER_VER</td>
</tr>
<tr>
<td>8</td>
<td>SQL_FETCH_DIRECTION</td>
</tr>
<tr>
<td>9</td>
<td>SQL_ODBC_API_CONFORMANCE</td>
</tr>
<tr>
<td>10</td>
<td>SQL_ODBC_VER</td>
</tr>
<tr>
<td>11</td>
<td>SQL_ROW_UPDATES</td>
</tr>
<tr>
<td>12</td>
<td>SQL_ODBC_SAG_CLI_CONFORMANCE</td>
</tr>
<tr>
<td>13</td>
<td>SQL_SERVER_NAME</td>
</tr>
<tr>
<td>14</td>
<td>SQL_SEARCH_PATTERN_ESCAPE</td>
</tr>
<tr>
<td>15</td>
<td>SQL_ODBC_SQL_CONFORMANCE</td>
</tr>
<tr>
<td>17</td>
<td>SQL_DBMS_NAME</td>
</tr>
<tr>
<td>18</td>
<td>SQL_DBMS_VER</td>
</tr>
<tr>
<td>19</td>
<td>SQL_ACCESSIBLE_TABLES</td>
</tr>
<tr>
<td>20</td>
<td>SQL_ACCESSIBLE_PROCEDURES</td>
</tr>
<tr>
<td>21</td>
<td>SQL_PROCEDURES</td>
</tr>
<tr>
<td>22</td>
<td>SQL_CONCAT_NULL_BEHAVIOR</td>
</tr>
<tr>
<td>23</td>
<td>SQL_CURSOR_COMMIT_BEHAVIOR</td>
</tr>
<tr>
<td>24</td>
<td>SQL_CURSOR_ROLLBACK_BEHAVIOR</td>
</tr>
<tr>
<td>25</td>
<td>SQL_DATA_SOURCE_READ_ONLY</td>
</tr>
<tr>
<td>26</td>
<td>SQL_DEFAULT_TXN_ISOLATION</td>
</tr>
<tr>
<td>27</td>
<td>SQL_EXPRESSIONS_IN_ORDERBY</td>
</tr>
<tr>
<td>28</td>
<td>SQL_IDENTIFIER_CASE</td>
</tr>
<tr>
<td>29</td>
<td>SQL_IDENTIFIER_QUOTE_CHAR</td>
</tr>
<tr>
<td>30</td>
<td>SQL_MAX_COLUMN_NAME_LEN</td>
</tr>
<tr>
<td>31</td>
<td>SQL_MAX_CURSOR_NAME_LEN</td>
</tr>
<tr>
<td>32</td>
<td>SQL_MAX_SCHEMA_NAME_LEN</td>
</tr>
<tr>
<td>32</td>
<td>SQL_MAX_OWNER_NAME_LEN</td>
</tr>
<tr>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>-------</td>
<td>-------------------------------------------------</td>
</tr>
<tr>
<td>33</td>
<td>SQL_MAX_PROCEDURE_NAME_LEN</td>
</tr>
<tr>
<td>34</td>
<td>SQL_MAX_CATALOG_NAME_LEN</td>
</tr>
<tr>
<td>34</td>
<td>SQL_MAX_QUALIFIER_NAME_LEN</td>
</tr>
<tr>
<td>35</td>
<td>SQL_MAX_TABLE_NAME_LEN</td>
</tr>
<tr>
<td>36</td>
<td>SQL_MULT_RESULT_SETS</td>
</tr>
<tr>
<td>37</td>
<td>SQL_MULTIPLE_ACTIVE_TXN</td>
</tr>
<tr>
<td>38</td>
<td>SQL_OUTER_JOINS</td>
</tr>
<tr>
<td>39</td>
<td>SQL_OWNER_TERM</td>
</tr>
<tr>
<td>40</td>
<td>SQL_PROCEDURE_TERM</td>
</tr>
<tr>
<td>41</td>
<td>SQL_QUALIFIER_NAME_SEPARATOR</td>
</tr>
<tr>
<td>42</td>
<td>SQL_QUALIFIER_TERM</td>
</tr>
<tr>
<td>43</td>
<td>SQL_SCROLL_CONCURRENCY</td>
</tr>
<tr>
<td>44</td>
<td>SQL_SCROLL_OPTIONS</td>
</tr>
<tr>
<td>45</td>
<td>SQL_TABLE_TERM</td>
</tr>
<tr>
<td>46</td>
<td>SQL_TXN_CAPABLE</td>
</tr>
<tr>
<td>47</td>
<td>SQL_USER_NAME</td>
</tr>
<tr>
<td>48</td>
<td>SQL_CONVERT_FUNCTIONS</td>
</tr>
<tr>
<td>49</td>
<td>SQL_NUMERIC_FUNCTIONS</td>
</tr>
<tr>
<td>50</td>
<td>SQL_STRING_FUNCTIONS</td>
</tr>
<tr>
<td>51</td>
<td>SQL_SYSTEM_FUNCTIONS</td>
</tr>
<tr>
<td>52</td>
<td>SQL_TIMEDATE_FUNCTIONS</td>
</tr>
<tr>
<td>53</td>
<td>SQL_CONVERT_BIGINT</td>
</tr>
<tr>
<td>54</td>
<td>SQL_CONVERT_BINARY</td>
</tr>
<tr>
<td>55</td>
<td>SQL_CONVERT_BIT</td>
</tr>
<tr>
<td>56</td>
<td>SQL_CONVERT_CHAR</td>
</tr>
<tr>
<td>57</td>
<td>SQL_CONVERT_DATE</td>
</tr>
<tr>
<td>58</td>
<td>SQL_CONVERT_DECIMAL</td>
</tr>
<tr>
<td>59</td>
<td>SQL_CONVERT_DOUBLE</td>
</tr>
<tr>
<td>60</td>
<td>SQL_CONVERT_FLOAT</td>
</tr>
<tr>
<td>61</td>
<td>SQL_CONVERT_INTEGER</td>
</tr>
<tr>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>-------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>62</td>
<td>SQL_CONVERT_LONGVARCHAR</td>
</tr>
<tr>
<td>63</td>
<td>SQL_CONVERT_NUMERIC</td>
</tr>
<tr>
<td>64</td>
<td>SQL_CONVERT_REAL</td>
</tr>
<tr>
<td>65</td>
<td>SQL_CONVERT_SMALLINT</td>
</tr>
<tr>
<td>66</td>
<td>SQL_CONVERT_TIME</td>
</tr>
<tr>
<td>67</td>
<td>SQL_CONVERT_TIMESTAMP</td>
</tr>
<tr>
<td>68</td>
<td>SQL_CONVERT_TINYINT</td>
</tr>
<tr>
<td>69</td>
<td>SQL_CONVERT_VARBINARY</td>
</tr>
<tr>
<td>70</td>
<td>SQL_CONVERT_VARCHAR</td>
</tr>
<tr>
<td>71</td>
<td>SQL_CONVERT_LONGVARBINARY</td>
</tr>
<tr>
<td>72</td>
<td>SQL_TXN_ISOLATION_OPTION</td>
</tr>
<tr>
<td>73</td>
<td>SQL_INTEGRITY</td>
</tr>
<tr>
<td>73</td>
<td>SQL_ODBC_SQL_OPT_IEF</td>
</tr>
<tr>
<td>74</td>
<td>SQL_CORRELATION_NAME</td>
</tr>
<tr>
<td>75</td>
<td>SQL_NON_NULLABLE_COLUMNS</td>
</tr>
<tr>
<td>76</td>
<td>SQL_DRIVER_HLIB</td>
</tr>
<tr>
<td>77</td>
<td>SQL_DRIVER_ODBC_VER</td>
</tr>
<tr>
<td>78</td>
<td>SQL_LOCK_TYPES</td>
</tr>
<tr>
<td>79</td>
<td>SQL_POS_OPERATIONS</td>
</tr>
<tr>
<td>80</td>
<td>SQL_POSITIONED_STATEMENTS</td>
</tr>
<tr>
<td>81</td>
<td>SQL_GETDATA_EXTENSIONS</td>
</tr>
<tr>
<td>82</td>
<td>SQL_BOOKMARK_PERSISTENCE</td>
</tr>
<tr>
<td>83</td>
<td>SQL_STATIC_SENSITIVITY</td>
</tr>
<tr>
<td>84</td>
<td>SQL_FILE_USAGE</td>
</tr>
<tr>
<td>85</td>
<td>SQL_NULL_COLLATION</td>
</tr>
<tr>
<td>86</td>
<td>SQL_ALTER_TABLE</td>
</tr>
<tr>
<td>87</td>
<td>SQL_COLUMN_ALIAS</td>
</tr>
<tr>
<td>88</td>
<td>SQL_GROUP_BY</td>
</tr>
<tr>
<td>89</td>
<td>SQL_KEYWORDS</td>
</tr>
<tr>
<td>90</td>
<td>SQL_ORDER_BY_COLUMNS_IN_SELECT</td>
</tr>
<tr>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>-------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>91</td>
<td>SQL_OWNER_USAGE</td>
</tr>
<tr>
<td>92</td>
<td>SQL_QUALIFIER_USAGE</td>
</tr>
<tr>
<td>93</td>
<td>SQL_QUOTED_IDENTIFIER_CASE</td>
</tr>
<tr>
<td>94</td>
<td>SQL_SPECIAL_CHARACTERS</td>
</tr>
<tr>
<td>95</td>
<td>SQL_SUBQUERIES</td>
</tr>
<tr>
<td>96</td>
<td>SQL_UNION</td>
</tr>
<tr>
<td>97</td>
<td>SQL_MAX_COLUMNS_IN_GROUP_BY</td>
</tr>
<tr>
<td>98</td>
<td>SQL_MAX_COLUMNS_IN_INDEX</td>
</tr>
<tr>
<td>99</td>
<td>SQL_MAX_COLUMNS_IN_ORDER_BY</td>
</tr>
<tr>
<td>100</td>
<td>SQL_MAX_COLUMNS_IN_SELECT</td>
</tr>
<tr>
<td>101</td>
<td>SQL_MAX_COLUMNS_IN_TABLE</td>
</tr>
<tr>
<td>102</td>
<td>SQL_MAX_INDEX_SIZE</td>
</tr>
<tr>
<td>103</td>
<td>SQL_MAX_ROW_SIZE_INCLUDES_LONG</td>
</tr>
<tr>
<td>104</td>
<td>SQL_MAX_ROW_SIZE</td>
</tr>
<tr>
<td>105</td>
<td>SQL_MAX_STATEMENT_LEN</td>
</tr>
<tr>
<td>106</td>
<td>SQL_MAX_TABLES_IN_SELECT</td>
</tr>
<tr>
<td>107</td>
<td>SQL_MAX_USER_NAME_LEN</td>
</tr>
<tr>
<td>108</td>
<td>SQL_MAX_CHAR_LITERAL_LEN</td>
</tr>
<tr>
<td>109</td>
<td>SQL_TIMEDATE_ADD_INTERVALS</td>
</tr>
<tr>
<td>110</td>
<td>SQL_TIMEDATE_DIFF_INTERVALS</td>
</tr>
<tr>
<td>111</td>
<td>SQL_NEED_LONG_DATA_LEN</td>
</tr>
<tr>
<td>112</td>
<td>SQL_MAX_BINARY_LITERAL_LEN</td>
</tr>
<tr>
<td>113</td>
<td>SQL_LIKE_ESCAPE_CLAUSE</td>
</tr>
<tr>
<td>114</td>
<td>SQL_QUALIFIER_LOCATION</td>
</tr>
<tr>
<td>115</td>
<td>SQL_OJ_CAPABILITIES</td>
</tr>
<tr>
<td>116</td>
<td>SQL_ACTIVE_ENVIRONMENTS</td>
</tr>
<tr>
<td>117</td>
<td>SQL_ALTER_DOMAIN</td>
</tr>
<tr>
<td>118</td>
<td>SQL_SQL_CONFORMANCE</td>
</tr>
<tr>
<td>119</td>
<td>SQL_DATETIME_LITERALS</td>
</tr>
<tr>
<td>120</td>
<td>SQL_BATCH_ROW_COUNT</td>
</tr>
<tr>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>121</td>
<td>SQL_BATCH_SUPPORT</td>
</tr>
<tr>
<td>122</td>
<td>SQL_CONVERT_WCHAR</td>
</tr>
<tr>
<td>123</td>
<td>SQL_CONVERT_INTERVAL_DAY_TIME</td>
</tr>
<tr>
<td>124</td>
<td>SQL_CONVERT_INTERVAL_YEAR_MONTH</td>
</tr>
<tr>
<td>125</td>
<td>SQL_CONVERT_WLONGVARCHAR</td>
</tr>
<tr>
<td>126</td>
<td>SQL_CONVERT_WVARCHAR</td>
</tr>
<tr>
<td>127</td>
<td>SQL_CREATE_ASSERTION</td>
</tr>
<tr>
<td>128</td>
<td>SQL_CREATE_CHARACTER_SET</td>
</tr>
<tr>
<td>129</td>
<td>SQL_CREATE_COLLATION</td>
</tr>
<tr>
<td>130</td>
<td>SQL_CREATE_DOMAIN</td>
</tr>
<tr>
<td>131</td>
<td>SQL_CREATE_SCHEMA</td>
</tr>
<tr>
<td>132</td>
<td>SQL_CREATE_TABLE</td>
</tr>
<tr>
<td>133</td>
<td>SQL_CREATE_TRANSLATION</td>
</tr>
<tr>
<td>134</td>
<td>SQL_CREATE_VIEW</td>
</tr>
<tr>
<td>135</td>
<td>SQL_DRIVER_HDESC</td>
</tr>
<tr>
<td>136</td>
<td>SQL_DROP_ASSERTION</td>
</tr>
<tr>
<td>137</td>
<td>SQL_DROP_CHARACTER_SET</td>
</tr>
<tr>
<td>138</td>
<td>SQL_DROP_COLLATION</td>
</tr>
<tr>
<td>139</td>
<td>SQL_DROP_DOMAIN</td>
</tr>
<tr>
<td>140</td>
<td>SQL_DROP_SCHEMA</td>
</tr>
<tr>
<td>141</td>
<td>SQL_DROP_TABLE</td>
</tr>
<tr>
<td>142</td>
<td>SQL_DROP_TRANSLATION</td>
</tr>
<tr>
<td>143</td>
<td>SQL_DROP_VIEW</td>
</tr>
<tr>
<td>144</td>
<td>SQL_DYNAMIC_CURSOR_ATTRIBUTES1</td>
</tr>
<tr>
<td>145</td>
<td>SQL_DYNAMIC_CURSOR_ATTRIBUTES2</td>
</tr>
<tr>
<td>146</td>
<td>SQL_FORWARD_ONLY_CURSOR_ATTRIBUTES1</td>
</tr>
<tr>
<td>147</td>
<td>SQL_FORWARD_ONLY_CURSOR_ATTRIBUTES2</td>
</tr>
<tr>
<td>148</td>
<td>SQL_INDEX_KEYWORDS</td>
</tr>
<tr>
<td>149</td>
<td>SQL_INFO_SCHEMA_VIEWS</td>
</tr>
<tr>
<td>150</td>
<td>SQL_KEYSET_CURSOR_ATTRIBUTES1</td>
</tr>
<tr>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>---------------------------------------------------------------</td>
</tr>
<tr>
<td>151</td>
<td>SQL_KEYSET_CURSOR_ATTRIBUTES2</td>
</tr>
<tr>
<td>152</td>
<td>SQL_ODBC_INTERFACE_CONFORMANCE</td>
</tr>
<tr>
<td>153</td>
<td>SQL_PARAM_ARRAY_ROW_COUNTS</td>
</tr>
<tr>
<td>154</td>
<td>SQL_PARAM_ARRAY_SELECTS</td>
</tr>
<tr>
<td>155</td>
<td>SQL_SQL92_DATETIME_FUNCTIONS</td>
</tr>
<tr>
<td>156</td>
<td>SQL_SQL92_FOREIGN_KEY_DELETE_RULE</td>
</tr>
<tr>
<td>157</td>
<td>SQL_SQL92_FOREIGN_KEY_UPDATE_RULE</td>
</tr>
<tr>
<td>158</td>
<td>SQL_SQL92_GRANT</td>
</tr>
<tr>
<td>159</td>
<td>SQL_SQL92_NUMERIC_VALUE_FUNCTIONS</td>
</tr>
<tr>
<td>160</td>
<td>SQL_SQL92_PREDICATES</td>
</tr>
<tr>
<td>161</td>
<td>SQL_SQL92_RELATIONAL_JOIN_OPERATORS</td>
</tr>
<tr>
<td>162</td>
<td>SQL_SQL92_REVOKE</td>
</tr>
<tr>
<td>163</td>
<td>SQL_SQL92_ROW_VALUE_CONSTRUCTOR</td>
</tr>
<tr>
<td>164</td>
<td>SQL_SQL92_STRING_FUNCTIONS</td>
</tr>
<tr>
<td>165</td>
<td>SQL_SQL92_VALUE_EXPRESSIONS</td>
</tr>
<tr>
<td>166</td>
<td>SQL_STANDARD_CLI_CONFORMANCE</td>
</tr>
<tr>
<td>167</td>
<td>SQL_STATIC_CURSOR_ATTRIBUTES1</td>
</tr>
<tr>
<td>168</td>
<td>SQL_STATIC_CURSOR_ATTRIBUTES2</td>
</tr>
<tr>
<td>169</td>
<td>SQL_AGGREGATE_FUNCTIONS</td>
</tr>
<tr>
<td>170</td>
<td>SQL_DDL_INDEX</td>
</tr>
<tr>
<td>171</td>
<td>SQL_DM_VER</td>
</tr>
<tr>
<td>172</td>
<td>SQL_INSERT_STATEMENT</td>
</tr>
<tr>
<td>10000</td>
<td>SQL_XOPEN_CLI_YEAR</td>
</tr>
<tr>
<td>10001</td>
<td>SQL_CURSOR_SENSITIVITY</td>
</tr>
<tr>
<td>10002</td>
<td>SQL_DESCRIBE_PARAMETER</td>
</tr>
<tr>
<td>10003</td>
<td>SQL_CATALOG_NAME</td>
</tr>
<tr>
<td>10004</td>
<td>SQL_COLLATION_SEQ</td>
</tr>
<tr>
<td>10005</td>
<td>SQL_MAX_IDENTIFIER_LEN</td>
</tr>
<tr>
<td>10021</td>
<td>SQL_ASYNC_MODE</td>
</tr>
<tr>
<td>10022</td>
<td>SQL_MAX_ASYNC_CONCURRENT_STATEMENTS</td>
</tr>
</tbody>
</table>
tables Function

The `tables` function returns a list of tables for the data source. Unlike the standard function for returning tables for a data source, you can use the `tables` function to filter the returned table names by specifying the appropriate case-sensitive SQL mask characters for your data source. To filter the list of tables returned, use the `tableNameMask` parameter, and set it to the appropriate mask:

```
http://[svr]/[path]/data/sql_ds?rpc=tables&tableNameMask=%%cust%%
```

Note

The double percent signs (%%) are necessary in the above example because URL syntax uses % to signify an escaped character. The SQL_DS data source requires the table name syntax in the query to be %tablename%.

The `tables` function uses the parameters shown in the following table.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Sample Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>tableNameMask</code></td>
<td>(Optional) The mask you want to use to filter the returned table names.</td>
<td><code>tableNameMask = %Company%%</code></td>
</tr>
<tr>
<td><code>includeSystemTables</code></td>
<td>(Optional) Includes the names of system tables. Defaults to 0. Can be 0, false, no; or 1, true, yes.</td>
<td><code>includeSystemTables = 1</code></td>
</tr>
</tbody>
</table>

columns Function

This function returns a list of columns for a specific table. As with the `tables` function, you can filter the columns returned using a mask. The `columns` function uses the parameters shown in the following table.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Sample Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>columnNameMask</code></td>
<td>(Optional) The mask you want to use to filter the returned columns.</td>
<td><code>columnNameMask = %a%%</code></td>
</tr>
<tr>
<td><code>tableName</code></td>
<td>The name of the table from which you want to retrieve columns. Case-sensitive.</td>
<td><code>tableName = %customer%%</code></td>
</tr>
</tbody>
</table>

Example: Retrieving columns from a table.

```
http://[svr]/[path]/netportal.dll/data/sql_ds?rpc=columns&tableName = %customer%% &format = html
```
columnPrivileges Function
This function returns a list of columns and associated privileges for a specified table. Some ODBC drivers do not support this function. The columnPrivileges function takes the same parameters as the columns function.

foreignKeys Function
You can obtain a list of foreign keys for a specific table, or find out how the primary key of one table is related to another table using the foreignKeys function. Some drivers may not support this function. The function uses the parameters shown on the following table.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Sample Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>primaryKeyTableName</td>
<td>The name of the table containing the primary key you want to inspect. Case-sensitive.</td>
<td>primaryKeyTableName = a</td>
</tr>
<tr>
<td>foreignKeyTableName</td>
<td>(Optional.) The name of the table you want to check for the foreign key relationship. Case-sensitive.</td>
<td>foreignKeyTableName = b</td>
</tr>
</tbody>
</table>

primaryKeys Function
The primaryKeys function returns a list of fields that make up the primary key. The only parameter is tableName, which is required and case-sensitive.

procedures Function
The procedures function returns a list of the stored procedures in a database. The only parameter is procNameMask, which is optional and case-sensitive.

procedureColumns Function
The procedureColumns function returns all input and output parameters for the specified stored procedure. Use the case-sensitive procName parameter to specify the stored procedure name. The columnNameMask parameter is optional, and is useful only if the ODBC driver returns column information for the procedure in addition to input and output parameter information. Use %% as a wildcard.

statistics Function
The statistics function retrieves a list of statistics about a single table and the indexes associated with that table. The only parameter is tableName, which is required and case-sensitive.
**getTipoInfo Function**

The `getTypeInfo` function returns information about data types supported by the data source. The only parameter is `odbcDataType`, which is an integer, and represents one of the values shown in the following table.

**Values for the odbcDataType Parameter**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>PN_SQL_LONGVARCHAR</td>
</tr>
<tr>
<td>-2</td>
<td>PN_SQL_BINARY</td>
</tr>
<tr>
<td>-3</td>
<td>PN_SQL_VARBINARY</td>
</tr>
<tr>
<td>-4</td>
<td>PN_SQL_LONGVARBINARY</td>
</tr>
<tr>
<td>-5</td>
<td>PN_SQL_BIGINT</td>
</tr>
<tr>
<td>-6</td>
<td>PN_SQL_TINYINT</td>
</tr>
<tr>
<td>-7</td>
<td>PN_SQL_BIT</td>
</tr>
<tr>
<td>0</td>
<td>PN_SQL_ALL_TYPES</td>
</tr>
<tr>
<td>1</td>
<td>PN_SQL_CHAR</td>
</tr>
<tr>
<td>2</td>
<td>PN_SQL_NUMERIC</td>
</tr>
<tr>
<td>3</td>
<td>PN_SQL_DECIMAL</td>
</tr>
<tr>
<td>4</td>
<td>PN_SQL_INTEGER</td>
</tr>
<tr>
<td>5</td>
<td>PN_SQL_SMALLINT</td>
</tr>
<tr>
<td>6</td>
<td>PN_SQL_FLOAT</td>
</tr>
<tr>
<td>7</td>
<td>PN_SQL_REAL</td>
</tr>
<tr>
<td>8</td>
<td>PN_SQL_DOUBLE</td>
</tr>
<tr>
<td>9</td>
<td>PN_SQL_DATETIME</td>
</tr>
<tr>
<td>10</td>
<td>PN_SQL_TIME</td>
</tr>
<tr>
<td>11</td>
<td>PN_SQL_TIMESTAMP</td>
</tr>
<tr>
<td>12</td>
<td>PN_SQL_VARCHAR</td>
</tr>
<tr>
<td>91</td>
<td>PN_SQL_TYPE_DATE</td>
</tr>
<tr>
<td>92</td>
<td>PN_SQL_TYPE_TIME</td>
</tr>
<tr>
<td>93</td>
<td>PN_SQL_TYPE_TIMESTAMP</td>
</tr>
</tbody>
</table>
Advanced Tag-based Metadata

You can use the following functions to request advanced metadata from tag-based data sources.

Functions for Requesting Advanced Metadata

<table>
<thead>
<tr>
<th>Query Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tagSearch</td>
<td>Returns a list of tags for the data source.</td>
</tr>
<tr>
<td>getSupportedAttributes</td>
<td>Returns a list of tag attributes supported by the data source.</td>
</tr>
<tr>
<td>tagAttributes</td>
<td>Returns the requested attributes for a given tag.</td>
</tr>
</tbody>
</table>

**tagSearch Function**

The `tagSearch` function returns a list of tags from a tag-based data source. It does not provide a list of tags for relational databases configured to appear as tag-based data sources.

**FMI**

For more information about using the `tagSearch` function, see “Retrieving a List of Tags” on page 39.

**getSupportedAttributes Function**

The `getSupportedAttributes` function provides a list of tag attributes that a particular data source supports. This function takes an optional `intfType` parameter that serves the same purpose as in the `tagSearch` function.

**FMI**

For more information about the `intfType` parameter, see “Parameters for Retrieving a List of Tags” on page 39.

Example: Retrieving a list of supported data source attributes.

```
http://[svr]/[path]/netportal.dll/data/opc_sim?rpc=getSupportedAttributes&format=html
```

**tagAttributes Function**

The `tagAttributes` function returns the values for a set of requested attributes for one or more tags. The valid attributes for the data source can be retrieved using the `getSupportedAttributes` function. The `tagAttributes` function uses the parameters shown in the following table.
Parameters for tagAttributes Function

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Sample Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>tag##</td>
<td>The tags from which you want to get attributes, where ## is the tag number, starting with 0.</td>
<td>tag0 = random.int2&amp;tag1 = random.int1</td>
</tr>
<tr>
<td>attribute##</td>
<td>The attributes you want to retrieve, where ## is the attribute number, starting with 0. The value of the attribute## is an integer representing the attribute type. These integers can be seen by using the getSupportedAttributes function.</td>
<td>attribute0 = 2</td>
</tr>
<tr>
<td>intfType</td>
<td>(Optional) Applies to OPC data sources. The interface type to use for doing tag searches, either real-time or historical (0 or 1). Defaults to real-time.</td>
<td>intfType = 1</td>
</tr>
</tbody>
</table>

Example: Retrieving a tag’s description.

http://[svr]/[path]/netportal.dll/data/opc_sim?tag0=random.int2&tag1=random.int1&attribute0=2&rpc=tagAttributes&format=html

Operational Insight System Data Access

In addition to retrieving information from various databases, you can also use NetPortal to obtain certain Operational Insight system-specific information. The syntax for getting access to this information is similar to the syntax for retrieving database data, except that an actual data source name is not required. Global variable values can be retrieved using the query types shown in the following table.

Query Types for Retrieving Global Variable Values

<table>
<thead>
<tr>
<th>Query Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getPNVariables</td>
<td>Retrieves Operational Insight global variable values.</td>
</tr>
<tr>
<td>getFunctionPermissions</td>
<td>Retrieves information about whether or not a user has access to specific Operational Insight functions.</td>
</tr>
</tbody>
</table>
Using getPNVariables

Operational Insight global variables are string values which can be configured within the NetManage application. These variables are grouped into categories, and can be retrieved individually, or as part of the entire category.

Global variables cannot be modified outside of the NetManage application, but can be accessed through the NetPortal’s getPNVariables query type, which returns a record set containing two columns: VariableName and Value. The data request for global variables uses the parameters shown in the following table.

Parameters for Requesting Global Variable Data

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Sample Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>category</td>
<td>Defines the category from which the variable values are taken.</td>
<td>category = General</td>
</tr>
<tr>
<td>variable0...variable99</td>
<td>(Optional) Specifies variables for which you want to retrieve values. If a variable parameter is not specified, values for all variables within the category are returned.</td>
<td>variable0 = AdminEmail</td>
</tr>
</tbody>
</table>

Note

The category “All” is available in Operational Insight 3.2 and subsequent versions.

Example: Accessing the AdminEmail and DateFormat global variables through a URL query.

http://[server]/path/Netportal/Netportal.dll/data/?category=general&variable0=AdminEmail&variable1=DateFormat&rpc=getPNVariables&format=html

In this example, no data source name is required. If one is specified, it is ignored. This query returns an HTML table (because of the HTML format specifier) similar to the following table.

AdminEmail and DateFormat Global Variables

<table>
<thead>
<tr>
<th>VariableName</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>AdminEmail</td>
<td><a href="mailto:admin@mycompany.com">admin@mycompany.com</a></td>
</tr>
<tr>
<td>DateFormat</td>
<td>mm/dd/yyyy HH:MM:SS</td>
</tr>
</tbody>
</table>

Example: Retrieving all global variables in the General category using the WebDataSet and JavaScript.
<HTML>
<HEAD>
<SCRIPT language = 'JavaScript'>
//Data access path. No data source name is required.
var sPath = "/ProcessNet/isapi/.netportal/netportal.dll/data/";
var sQuery = "category = general&rpc = getPNVariables";
var rsResults = PNWebDS.openRecordSet(sPath, sQuery);
//Get the returned record set fields.
var fldVarName = rsResults.FieldByName("VariableName");
var fldValue = rsResults.FieldByName("Value");
//Build an associative array of the variable results.
var aGlobalVariables = new Array();
//Iterate through the fields, populating the array.
while(rsResults.Eof() == false)
{
    aGlobalVariables[fldVarName.AsString()] = fldValue.AsString();
    //Advance to the next record.
    rsResults.Next();
}
//Access global variable from associative array.
var sAdminEmail = aGlobalVariables["AdminEmail"];
</SCRIPT>
...
</HTML>

FMI
This example assumes that a WebDataSet object with an ID of PNWebDS has been defined in the HTML document. For more information about using the WebDataSet, see “Using the WebDataSet” on page 95.

Using getFunctionPermissions

Operational Insight uses objects, called functions, to control various permission settings. Each function is associated with a key value, which behaves as a category, grouping functions within it. You can retrieve information about whether the currently logged on user has permission to use these functions through the NetPortal’s getFunctionPermissions query type, which returns a record set containing two fields: Function, a string containing the function name, and Permissions, a Boolean value that indicates the current user’s permission to use the function. A query for function permissions uses the parameters shown in the following table.
Parameters for `getFunctionPermissions` Query

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Sample Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>key</code></td>
<td>The key category from which the function permissions are retrieved.</td>
<td><code>key = Generic</code></td>
</tr>
<tr>
<td><code>function0...</code></td>
<td>(Optional) Specifies functions for which to retrieve permissions. If a function parameter is not specified, permissions for all functions associated with the key are returned.</td>
<td><code>function0 = AddSQL</code></td>
</tr>
</tbody>
</table>

Example: Accessing permissions for all functions of the `Generic` key category.

```
http://[server]/[path]/NetPortal/NetPortal.dll/data/?key=Generic&rpc=getFunctionPermissions&format=html
```

In this example, no data source name is required. If one is specified, it is ignored.

Note that this returns only those functions that have permissions set to true unless the function is explicitly requested.

This query returns an HTML table (because of the HTML format specifier) similar to the following table.

**HTML Table Returned by `getFunctionPermissions` Query**

<table>
<thead>
<tr>
<th>Function</th>
<th>Permission</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChangePassword</td>
<td>True</td>
</tr>
<tr>
<td>AddSQL</td>
<td>True</td>
</tr>
</tbody>
</table>

**Other NetPortal Functions**

Operational Insight provides additional functions for session management and content access.

**Log On, Log Off**

Logging on to Operational Insight can be done in two ways:

1. By displaying the logon page.
2. By submitting the logon page information directly to Netportal through the URL.

Logging off can also be done in two ways:

1. By using the `LogOut` function.
2. By using the `closeSession` RPC (Remote Procedure Call).
Displaying the Logon Page

There are several parameters that can be used in the path which results in the logon page being displayed. These are shown in the following table.

Parameters for Displaying the Logon Page

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Sample Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>redirect</td>
<td>(Optional) Specifies the text that will replace the <code>&lt;#Redirect&gt;</code> token in the logon page template.</td>
<td>redirect = /ProcessNet/html/main.html</td>
</tr>
<tr>
<td>Err</td>
<td>(Optional) Specifies the text that will replace the <code>&lt;#ErrorMessage&gt;</code> token in the logon page template.</td>
<td>err = MyError</td>
</tr>
</tbody>
</table>

Example: Displaying the logon page.

http://[svr]/[path]/netportal.dll/loginpage

Submitting Logon Information Through the URL

Logon information that would usually be entered in the logon page can be submitted to the NetPortal by using the SubmitLogin function in the command string of the URL.

**Caution**

This direct method is not recommended, since it exposes the user name and password in plain text. A safer way of using this method to submit logon information is shown later in this section, using a generic guest user name and password. For more information about alternatives, see “HTML Forms” on page 67.

Example: Submitting logon information through a URL.

http://[svr]/[path]/netportal.dll/
SubmitLogin?username=guestuser&password=matrikon

The SubmitLogin function takes several parameters, as shown in the following table.
Chapter 3: NetPortal Data Access Interface

Parameters for SubmitLogin Function

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Sample Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>useIL</td>
<td>(Optional) When specified, an integrated login using the current NT user is attempted (username, password, explicitlogin, and timezone are available when using this parameter).</td>
<td>useIL = true</td>
</tr>
<tr>
<td>username</td>
<td>The name of the user logging on.</td>
<td>username = GuestUser</td>
</tr>
<tr>
<td>password</td>
<td>The password for the user.</td>
<td>password = matrikon</td>
</tr>
<tr>
<td>timezone</td>
<td>(Optional) The client time zone in minutes from GMT.</td>
<td>timezone = 360</td>
</tr>
<tr>
<td>redirect</td>
<td>(Optional) The URL you want to load once the logon has been submitted.</td>
<td>redirect = /ProcessNet/ html/main.html</td>
</tr>
<tr>
<td>explicitlogin</td>
<td>(Optional) When the server is using integrated login, forces the specified user name and password to be used instead of the Windows user name and password. While opening multiple sessions to the Operational Insight server using the same username &amp; password, explicitlogin=true should be set, else every alternate attempts to connect will fail. This is applicable even when integrated login is not configured.</td>
<td>explicitlogin = true</td>
</tr>
</tbody>
</table>

The syntax used to submit a logon can be appended to any data request, so that there is an automatic log on to Operational Insight before the request is processed. This is especially useful when connecting to Operational Insight pages from external pages, for example, in the following scenario:

1. A plant has an intranet page on which users require a link to a page containing process data.
2. The Operational Insight administrator creates the page that contains the process data in NetDraw, and saves it so that users with access to the plant intranet can view the page.
3. The intranet page link to the NetDraw page should not force anyone to log on to Operational Insight in order to view the page, since some users will not use Operational Insight regularly.
In this case, the Operational Insight administrator can specify the link path so that it automatically logs the user on as a guest user when the NetDraw page is opened, and the page can be viewed through the browser.

Example: Submitting guest user logon information through a URL for page access.

http://[svr]/ProcessNet/html/
NetDrawPage?ConfigID=123&username=guestuser&password=matrikon

**Note**

If integrated login is being used, `explicitlogin=true` must be added to the above example in order to force the use of the specified user name and password.

While opening multiple sessions to the Operational Insight server using the same username & password, `explicitlogin=true` should be set, else every alternate attempts to connect will fail. This is applicable even when integrated login is not configured.

---

**Logging Off Using the LogOut Function**

The currently logged on user can be logged off by using the LogOut function. By default, the user is redirected to the main logon page. Use the `Redirect` parameter to redirect the user to a different page once they are logged out.

Example: Logging off using LogOut.

http://[svr]/[path]/netportal.dll/LogOut

**Logging Off Using the closeSession Call**

The `closeSession` RPC (Remote Procedure Call) can also be used to log off. Any text can be used as the data source name (in the following example, `PNQueryDB`; it is ignored, but must be included as a placeholder to create a well-formed URL.

Example: Logging off using closeSession.

http://[svr]/[path]/netportal.dll/data/PNQueryDB?rpc=closeSession

**Note**

The `closeSession` RPC is available for use in Operational Insight 3.2 and subsequent versions.

---

**Change Password**

Users can change their own passwords using the ChangePassword function.
Caution
This direct method is not recommended, since it exposes the user name and password in plain text. For more information about alternatives, see “HTML Forms” on page 67.

Example: Changing the password using a URL.

http://[svr]/[path]/netportal.dll/ChangePassword?password=matrikon&new=Nokirtam&confirm=Nokirtam

The ChangePassword function takes several parameters, as shown in the following table.

Parameters for ChangePassword Function

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Sample Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>password</td>
<td>(Optional) The password for the user.</td>
<td>password = matrikon</td>
</tr>
<tr>
<td>new</td>
<td>(Optional) The new password.</td>
<td>new = Nokirtam</td>
</tr>
<tr>
<td>confirm</td>
<td>(Optional) The new password, re-entered for confirmation.</td>
<td>confirm = Nokirtam</td>
</tr>
<tr>
<td>username</td>
<td>(Optional) The user name.</td>
<td>username = jane.doe</td>
</tr>
<tr>
<td>explicitlogin</td>
<td>(Optional) When the server is using integrated login, forces the specified user name and password to be used instead of the Windows user name and password. While opening multiple sessions to the Operational Insight server using the same username &amp; password, explicitlogin=true should be set, else every alternate attempts to connect will fail. This is applicable even when integrated login is not configured.</td>
<td>explicitlogin = true</td>
</tr>
<tr>
<td>timezone</td>
<td>(Optional) The client timezone in minutes from GMT.</td>
<td>timezone = 360</td>
</tr>
</tbody>
</table>
Session Information

The **Info** function provides session information for the current user session, including the time zone and whether or not the session is active. The only parameter, **RecordSet**, is optional. If true, session information is returned as a record set. The **format** parameter can be used along with **RecordSet** to specify the packet provider.

**Example: Getting session information.**

```
http://[svr]/[path]/netportal.dll/info
```

**Note**
The **RecordSet** parameter is available for use in Operational Insight 3.2 and subsequent versions.

Version Information

The version of the current server installation of Operational Insight can be acquired by using the **VersionInfo** function. The optional **format** parameter can be used along with **VersionInfo** to specify the return format. The default return format is binary.

**Example: Getting version information.**

```
http://[svr]/[path]/netportal.dll/VersionInfo?format=html
```

Ping Session

Using the **PingSession** function, you can determine the status of the current user session. The optional **format** parameter can be used along with **PingSession** to specify the return format. The default return format is binary.

### Parameters for PingSession Function

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Sample Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>format</td>
<td>(Optional) The format of the return data. If the format is not specified, the PingSession function will return binary data.</td>
<td>format = html</td>
</tr>
<tr>
<td>password</td>
<td>(Optional) The password for the user.</td>
<td>password = matrikon</td>
</tr>
<tr>
<td>username</td>
<td>(Optional) The user name.</td>
<td>username = jane.doe</td>
</tr>
</tbody>
</table>
### Chapter 3: NetPortal Data Access Interface

#### Parameter Description Sample Usage

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Sample Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>timezone</td>
<td>(Optional) The client timezone in minutes from GMT.</td>
<td>timezone = 360</td>
</tr>
<tr>
<td>explicitlogin</td>
<td>(Optional) When the server is using integrated login, forces the specified user name and password to be used instead of the Windows user name and password. While opening multiple sessions to the Operational Insight server using the same username &amp; password, explicitlogin=true should be set, else every alternate attempts to connect will fail. This is applicable even when integrated login is not configured.</td>
<td>explicitlogin = true</td>
</tr>
</tbody>
</table>

Example: Checking session status.
```
http://[svr]/[path]/netportal.dll/PingSession?format=html
```

**HTML/Operational Insight Content**

Operational Insight content, such as trends or process graphics, is stored in the Operational Insight content database. This content is accessible to Web clients through the NetPortal.

Example: Accessing Operational Insight content by name. This example retrieves a saved trend called **Example Trend**.
```
http://[svr]/[path]/netportal.dll/html/Example Trend
```
Parameters for Operational Insight Content Access

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Sample Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConfigID</td>
<td>(Optional) The ID number of the content.</td>
<td>ConfigID = 123</td>
</tr>
<tr>
<td>username</td>
<td>(Optional) The user name.</td>
<td>username = jane.doe</td>
</tr>
<tr>
<td>password</td>
<td>(Optional) The password for the user.</td>
<td>password = matrikon</td>
</tr>
<tr>
<td>timezone</td>
<td>(Optional) The client timezone in minutes from GMT.</td>
<td>timezone = 360</td>
</tr>
<tr>
<td>explicitlogin</td>
<td>(Optional) When the server is using integrated login, forces the specified user name and password to be used instead of the Windows user name and password. While opening multiple sessions to the Operational Insight server using the same username &amp; password, explicitlogin=true should be set, else every alternate attempts to connect will fail. This is applicable even when integrated login is not configured.</td>
<td>explicitlogin = true</td>
</tr>
</tbody>
</table>

Understanding Database Content

Getting access to Operational Insight content using the name of the object can have unexpected results, because there may be multiple items with the same name. This is because different users can save content, such as trends, with the same name. However, all items in the content database are assigned a unique ID, called the ConfigID. Using the ConfigID is the preferred method for getting access to content from Operational Insight.

As an option, you can store your HTML, PNSP, and other content in the content database. This enables you to apply security to your content from the NetManage administrator utility. You can also view statistics on the users that access your content.

Example: Accessing Operational Insight content by ID.

http://[svr]/[path]/netportal.dll/html?ConfigID=123
The ConfigID for content can be determined in several ways:

- By using NetManage.
- By using a URL query against PNQueryDB, with the appropriate SQL statement.
- By adding a link to the item in the profile tree, enabling viewing of the link's properties through the context menu.

**Note**
The ConfigID for content will not be the same if the content is added to the content database of a different Operational Insight server. For this reason, content containing links to other content by ConfigID is not portable to another Operational Insight server without modifications.

**FMI**
For more information about using NetManage to apply security to content, view statistics on content use, or determine a ConfigID, see "Using the NetManage Administrator Console" in the *Operational Insight Administrator’s Guide*.

### NetPortal Data Access Function Summary

A summary of all the RPCs and functions, including their parameters, is shown in the following table. The square brackets indicate that a parameter is optional.

**Summary of NetPortal Remote Procedure Calls and Functions**

<table>
<thead>
<tr>
<th>RPC</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>getSupportedAttributes</td>
<td>[intfType]</td>
</tr>
<tr>
<td>getInfo</td>
<td>infoType</td>
</tr>
<tr>
<td>ColumnPrivileges</td>
<td>tableName,[ColumnNameMask]</td>
</tr>
<tr>
<td>Columns</td>
<td>tableName,[ColumnNameMask]</td>
</tr>
<tr>
<td>ForeignKeys</td>
<td>PrimaryKeyTableName, [ForeignKeyTableName]</td>
</tr>
<tr>
<td>PrimaryKeys</td>
<td>tableName</td>
</tr>
<tr>
<td>GetTypeInfo</td>
<td>[odbcDataType]</td>
</tr>
<tr>
<td>ProcedureColumns</td>
<td>procName,[ColumnNameMask]</td>
</tr>
<tr>
<td>Procedures</td>
<td>[procNameMask]</td>
</tr>
</tbody>
</table>
### RPC

<table>
<thead>
<tr>
<th>Function</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statistics</td>
<td>tableName</td>
</tr>
<tr>
<td>tables</td>
<td>[tableNameMask, includeSystemTables]</td>
</tr>
<tr>
<td>CloseSession</td>
<td>No parameters</td>
</tr>
<tr>
<td>tagAttributes</td>
<td>tag##, attribute##,[intfType]</td>
</tr>
<tr>
<td>tagSearch</td>
<td>tag0 or tagNameMask, tag1 or desc, maxRecords,[doHierarchal, folderPath, intfType]</td>
</tr>
<tr>
<td>rawTagHistory</td>
<td>tag or tag0,[start, end, timebounds, timezone, maxRecords]</td>
</tr>
<tr>
<td>ProcessedTagHistory</td>
<td>tag#, aggregate or function, [start, end, maxRecords, sampleInterval or step]</td>
</tr>
<tr>
<td>getPnVariables</td>
<td>category,[variable##]</td>
</tr>
<tr>
<td>getFunctionPermissions</td>
<td>key,[function##]</td>
</tr>
<tr>
<td>writeBlob</td>
<td>SQL, blobData, [format, dataEncoding, ignoreWhere]</td>
</tr>
</tbody>
</table>

### Function

<table>
<thead>
<tr>
<th>Function</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>LoginPage</td>
<td>[useIL, Redirect, Err, userName, Password, ExplicitLogin, Timezone]</td>
</tr>
<tr>
<td>SubmitLogin</td>
<td>[Redirect, userName, Password, ExplicitLogin, Timezone]</td>
</tr>
<tr>
<td>Info</td>
<td>[RecordSet, format]</td>
</tr>
<tr>
<td>data</td>
<td>No parameters</td>
</tr>
<tr>
<td>html</td>
<td>No parameters</td>
</tr>
<tr>
<td>ChangePassword</td>
<td>password, new, confirm, [userName, ExplicitLogin, Timezone]</td>
</tr>
<tr>
<td>LogOut</td>
<td>[redirect]</td>
</tr>
<tr>
<td>PingSession</td>
<td>[format, userName, Password, ExplicitLogin, Timezone]</td>
</tr>
<tr>
<td>VersionInfo</td>
<td>[format]</td>
</tr>
</tbody>
</table>
Chapter 4: HTML Forms

An HTML form is a collection of HTML objects designed for user data entry. You can use HTML forms to perform data access much like with the URL queries described earlier in this guide.

FMI

For more information about getting access to data using URL queries, see “URL Queries” on page 27.

HTML Form Objects

A number of elements are used to create forms in HTML. The most common are shown in the following table.

<table>
<thead>
<tr>
<th>Object Type</th>
<th>Element Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Form</td>
<td><code>&lt;FORM&gt;&lt;/FORM&gt;</code></td>
<td>Defines the form and the action to be taken when the form is submitted to the server.</td>
</tr>
<tr>
<td>Edit box</td>
<td><code>&lt;INPUT type = text&gt;</code></td>
<td>A simple text entry field.</td>
</tr>
<tr>
<td>Variable</td>
<td><code>&lt;INPUT type = hidden&gt;</code></td>
<td>A hidden field used to represent a parameter sent to the server.</td>
</tr>
<tr>
<td>Submit button</td>
<td><code>&lt;INPUT type = submit&gt;</code></td>
<td>A button that sends the form data to the server.</td>
</tr>
<tr>
<td>Reset button</td>
<td><code>&lt;INPUT type = reset&gt;</code></td>
<td>A button that resets the form to the initial values.</td>
</tr>
<tr>
<td>Combo box</td>
<td><code>&lt;SELECT&gt;</code> <code>&lt;OPTION&gt;...</code> <code>&lt;/SELECT&gt;</code></td>
<td>A drop-down list of values.</td>
</tr>
</tbody>
</table>
The Form Element

The form element contains all the individual input elements and defines how the data gets processed when the user clicks \texttt{Submit}. The form element has three main attributes, as shown in the following table.

<table>
<thead>
<tr>
<th>Object Type</th>
<th>Element Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>List box</td>
<td>&lt;SELECT size=2&gt;\n/OPTION&gt;...\n&lt;/SELECT&gt;, where size is the number of variables that will be displayed in the list box on the page.</td>
<td>A list of values.</td>
</tr>
<tr>
<td>Multi-line edit box</td>
<td>&lt;TEXTAREA&gt;&lt;/TEXTAREA&gt;</td>
<td>An edit box that can have multiple lines of text in it.</td>
</tr>
</tbody>
</table>

Form Element Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Sample Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>method</td>
<td>The HTTP method used to send the form to the server. This should always be set to \texttt{POST}.</td>
<td>method = post</td>
</tr>
<tr>
<td>action</td>
<td>The URL to which the form data is sent when the form is submitted.</td>
<td>action = /ProcessNet/\nnetportal/\nnetportal.dll/data/OPC_SIM</td>
</tr>
<tr>
<td>target</td>
<td>An optional attribute indicating which window or frame to use to display the results of the form submission.</td>
<td>target = DataWindow</td>
</tr>
</tbody>
</table>

The action is a URL that gets accessed when the form is submitted. When using forms with Operational Insight, the action attribute of a form normally references the NetPortal and a data source.

When the form is submitted, the equivalent of a URL query is created and sent to the server. A URL query uses an \texttt{HTTP GET} method, whereas a form normally uses an \texttt{HTTP POST} method. This means that a form is not restricted in the amount of data it can pass to the server, whereas a URL query is normally restricted to 256, 512, or 1024 bytes of data, depending on the server.
Forms for Data Access

You can use forms to create simple data query applications. By naming form elements with the name of the parameters that the NetPortal expects for data requests, you can eliminate URL queries.

Example: Accessing raw tag history using a form (Form.html).

```html
<HTML>
  <BODY>
    <FORM action="/ProcessNet/isapi/netportal/netportal.dll/data/opc_sim" method=post>
      Tagname: <INPUT name=tag0>
      Start Time: <INPUT name=start value='*-1d'>
      End Time: <INPUT name=end value='*'>
      Function: <INPUT name=function value=interp>
      Step: <INPUT name=step value='30M'>
      <INPUT type=hidden name=format value="HTML">
    </FORM>
  </BODY>
</HTML>
```

The preceding example uses the name attribute to identify the form objects. This name is sent to the server, which uses it to identify the data entered in the field. For example, entering *-1d in the start input box would result in the server receiving start = *-1d, exactly like a URL query.

Using Selection Lists

Users are often required to select values from a predefined list of values in applications. To generate these lists, use the `SELECT` element in combination with `OPTION` elements. `OPTION` elements represent the items in the list, and have both a value and text associated with them. The option selected by the user is then used as the value for the `SELECT` variable when the form is submitted.

Example: Using selection list boxes in forms (Selection.html).

```html
<HTML>
  <BODY>
    <FORM action="/ProcessNet/isapi/netportal/netportal.dll/data/opc_sim" method=post>
      Tagname: <INPUT name=tag0>
      Start Time: <INPUT name=start value='*-1d'>
      End Time: <INPUT name=end value='*'>
      Function: <INPUT name=function value=interp>
      Step: <INPUT name=step value='30M'>
      Format: <SELECT name=format>
```
Forms for Data Entry

When creating data entry forms, you must typically perform an additional step with some scripting to combine individual entry fields into a single SQL statement. The following example shows a simple entry form.

Example: Entering data into the industry table in the sample database with a form (Insert.html).

```
<HTML>
<HEAD>
  <SCRIPT>
    function doOnSubmit()
    {
      var sSQL = "insert into industry " + 
          "values (" + frmMain.edtCode.value + "," + 
          "" + frmMain.edtName.value + "," + "" + 
          frmMain.edtLong.value + "," + 
          frmMain.exec.value = sSQL;
    }
  </SCRIPT>
</HEAD>
<BODY>
  <FORM id=frmMain
    action="/ProcessNet/isapi/netportal/netportal.dll/
    data/SQL_DS"
    method=post onsubmit="doOnSubmit()"
    Code: <INPUT name=edtCode>
    <BR>
    Industry Name: <INPUT name=edtName>
    <BR>
    Long Name: <INPUT name=edtLong>
    <INPUT type=hidden name=exec>
    <BR>
    <INPUT type=submit>
    <INPUT type=hidden name=format value='HTML'>
  </FORM>
</BODY>
</HTML>
```

The previous example introduces two new concepts. The first is a naming technique called Hungarian Notation, which requires all variable and object names to be prefixed with the object type. In this example, frm is the notation for a form, and edt is the notation for edit boxes.

The second new concept is the use of a parent element to identify the form elements. For example, in the script, you access each input box by prefixing it with the parent form’s ID.
Chapter 5: Advanced PNSP

Previous sections dealt with the concept of using PNSP to generate HTML based on, or also combined with, data requests. PNSP is capable of more advanced tasks, as described in the following sections.

Generating Form Elements

PNSP is most useful when used to generate lists of selections, such as list boxes and combo boxes. However, PNSP can also be used to generate values for other form elements, such as edit boxes and text areas.

Note
You must save PNSP pages with a PNSP extension.

Example: Using PNSP to generate a drop-down list from a database (PNSPDropdown.pnsp).

```html
<HTML>
<HEAD>
  <SCRIPT defer=true>
    function frmMain.cbList.onchange()
    {
      alert("Industry code = " + frmMain.cbList.value);
    }
  </SCRIPT>
</HEAD>
<BODY>
  [#DataSet id=qryList
  sql="select IND_CODE, LONG_NAME from industry"
  datasource=SQL_DS]
  <FORM id=frmMain>
    <SELECT id=cbList size=1>
      [#Band datasource=qryList
      definition="<OPTION value=#Value Field=IND_CODE>[
      #Value Field=LONG_NAME]*"]
    </SELECT>
  </FORM>
</BODY>
</HTML>
```

If the form is not being submitted, such as when you are only using the form elements with client-side script, you do not need the enclosing `<FORM>` element.

Example: Using PNSP to generate a drop-down list using client-side data (PNSPSelect.pnsp).

```html
<HTML>
<HEAD>
  <SCRIPT defer=true>
    function cbList.onchange()
    {
      alert("Industry code = " + cbList.value);
    }
  </SCRIPT>
</HEAD>
<BODY>
  [#DataSet id=qryList
  sql="select IND_CODE, LONG_NAME from industry"
  datasource=SQL_DS]
  <FORM id=frmMain>
    <SELECT id=cbList size=1>
      [#Band datasource=qryList
      definition="<OPTION value=#Value Field=IND_CODE>[
      #Value Field=LONG_NAME]*"]
    </SELECT>
  </FORM>
</BODY>
</HTML>
```
To change the drop-down list to a list box, change the `size` attribute in the `<SELECT>` element to a value greater than 1.

Example: Using PNSP to fill a text edit box (`PNSPEdit.pnsp`).

```
<HTML>
  <BODY>
    [#DataSet id=qryText
      sql="select Notes, Category from biolife where Common_Name='Firefish'
      datasrc=SQL_DS]
    <TEXTAREA rows=20 cols=60>
      [#Value Field=0 datasrc=qryText]
    </TEXTAREA>
  </BODY>
</HTML>
```

Generating Client-Side Scripts

You can use PNSP to generate any text, including JavaScript or VBScript constructs, such as arrays or variables. When the PNSP file is executed, the generated script can be seen by right-clicking on the resulting page in the browser, and selecting View Source.

Example: Generating a JavaScript array with PNSP (`PNSPJavascript.pnsp`).

```
<SCRIPT>
  [#DataSet id=qryArray
    sql="select CustNo from customer" datasrc=SQL_DS]
  var aMyArray = new Array(
    [#Band datasrc=qryArray definition="[#Value field=0]," null);
</SCRIPT>
</BODY>
</HTML>
```

---

**Note**

PNSP is currently not capable of executing server-side scripts.
Using Dynamic Properties With PNSP

Dynamic properties allow you to set DHTML styles based on an expression that the Web browser evaluates when it displays the HTML page. Dynamic properties can be used with PNSP to change almost any visual property of the page, based on values generated by PNSP. For example, you could change the fill of an object, the color of an object’s text, or the position of an object dynamically.

Example: Using dynamic properties to highlight values out of range (PNSPProperties.pnsp).

```html
<HTML>
  <HEAD>
    <SCRIPT>
      function setColor(sText)
      {
        if(parseInt(sText)>50)
          return "red";
        return "black";
      }
    </SCRIPT>
  </HEAD>
  <BODY>
    <![Dataset id=qryData datasrc=OPC_Sim tags=".sinusoid" start=-10M end=* step=1M function=interp]>
      <TABLE>
        <TR>
          <TH>Time</TH>
          <TH>Value</TH>
        </TR>
        <![Band datasrc=qryData definition="<TR>
          <TD>[[Value Field=0]]</TD>
          <TD style="color:expression(setColor(this.innerText)) ">
            <![Value Field=1]]
          </TD>
        </TR>">
      </TABLE>
    </BODY>
  </HTML>
```

Master-Detail Bands

Using PNSP, you can embed Band and Dataset elements within other band definitions. You can use this technique to create master-detail reports.

Example: Using embedded band elements (PNSPMasterBand.pnsp).

```html
<HTML>
  <BODY>
    <![Dataset id=qryMaster datasrc=SQL_DS sql="select CustNo, Company from customer"]>
    <![Band datasrc=qryMaster definition="
      <![Dataset id=qryDetail datasrc=SQL_DS sql="select OrderNo, ItemsTotal from orders where CustNo=[#value datasrc=qryMaster Field=0]"]>
      <![Value Field=1]></B>
      <BR>
      <![Band datasrc=qryDetail definition="[Value Field=0]:
        \$[Value Field=1 Format=0.2]<BR>"
    ]"
  </BODY>
```
When embedding elements within band definitions, make sure you identify (escape) the double quotes in the embedded elements by placing a backslash (\) in front of them. You must also reference the proper data set when specifying value fields in an embedded data set.

**Generating Queries From PNSP**

Using PNSP, you can dynamically generate a tag-based query from a data set that returns the tag list.

**Generating Tag Lists**

You can use PNSP to generate a list of tags from a relational database to use in a tag-based query. To do this, you must create a data set element that retrieves the tag list, and a data set element that retrieves the tag values. In the tag data set element, instead of the list of tags, reference the data set that contains the list of tags using the notation:

```
src = dataset ID, field
```

Where `dataset ID` is the ID of the data set that contains the tag list, and `field` is the field within the data set that contains the tag names. The `field` can be a numerical index or a field name.

Example: Tag query based on a list of tags from a SQL database (PNSPTagList.pnsp).

```html
<HTML>
<BODY>
[#dataset id=qTags datasrc=sql_ds
    sql="select * FROM tags where tagname = '.sinusoid'"]
[#dataset id=qVals datasrc=OPC_SIM
    tags="src=qTags,TagName"]
[#Value datasrc=qVals field="Random.Int2.Value"]
<br>
[#Value datasrc=qVals field="Random.Int1.Value"]
</BODY>
</HTML>
```

**Note**

This example does not work unless you create a table called `Tags` in the SQL_DS database that has a field named `TagName` that contains the tag names.
Chapter 6: Operational Insight Messages and Dialog Boxes

Operational Insight uses several messages and application dialog boxes in its user interface. You can make use of these in your own applications. You can access all Operational Insight messages and dialog boxes by including the appropriate JavaScript files in your own pages. The following sections describe some of the messages and dialog boxes available in the system.

Tip
To see full implementation details for all of the available messages or dialog boxes, open PNDialogs.js in a text editor and view the comments.

Messages

The following table describes some of the messages available in Operational Insight. You can get access to all of these messages by including the PNDialogs.js file in your pages.

### Operational Insight Messages

<table>
<thead>
<tr>
<th>Dialog Box Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MessageDialog</td>
<td>Displays a message and selection buttons to the user.</td>
</tr>
<tr>
<td>AlertDialog</td>
<td>A dialog box with a warning icon and warning message.</td>
</tr>
<tr>
<td>PromptUser</td>
<td>Prompts the user for text.</td>
</tr>
<tr>
<td>ConfirmDialog</td>
<td>A dialog box with OK and Cancel buttons.</td>
</tr>
<tr>
<td>ErrorMessage</td>
<td>A dialog box with an error icon and an error message.</td>
</tr>
<tr>
<td>ExtendedAlertDialog</td>
<td>A dialog box with both simple and detailed error messages.</td>
</tr>
</tbody>
</table>

**MessageDialog**

The MessageDialog function displays a message and selection buttons to the user. It requires the parameters shown in the following table.
Chapter 6: Operational Insight Messages and Dialog Boxes

Parameters for MessageDialog Function

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sTitle</td>
<td>String</td>
<td>The title for the dialog box.</td>
</tr>
<tr>
<td>sPrompt</td>
<td>String</td>
<td>The prompt you want to be displayed to the user.</td>
</tr>
<tr>
<td>sButtons</td>
<td>String</td>
<td>A comma-separated list of buttons you want to be displayed.</td>
</tr>
<tr>
<td>sHelpURL</td>
<td>String, optional</td>
<td>The URL that contains help for the current prompt.</td>
</tr>
<tr>
<td>sIcon</td>
<td>String, optional</td>
<td>The URL for the icon you want to be displayed in the dialog box.</td>
</tr>
<tr>
<td>oWin</td>
<td>Window, optional</td>
<td>The parent window for the dialog box.</td>
</tr>
<tr>
<td>bModeless</td>
<td>Boolean, optional</td>
<td>Determines whether the dialog box is modal or modeless. The default is modal.</td>
</tr>
</tbody>
</table>

MessageDialog returns the text of the button clicked by the user.

Example: Using MessageDialog (MessageDialog.html).

```html
<html>
  <head>
    <script src="/ProcessNet/html/Dialogs/scr/PNDialogs.js"></script>
    <script defer=true>
      function btnDialog.onclick()
      {
        var sBtn = MessageDialog("Test Dialog", "This is a message", "Button1,Button2,Another One");
        alert(sBtn);
      }
    </script>
  </head>
  <body>
    <button id=btnDialog>Click Me</button>
  </body>
</html>
```

AlertDialog

The AlertDialog function provides a convenient way to display a message dialog box that contains a warning. Pass AlertDialog the message you want to display.

Example: Using AlertDialog (AlertDialog.html).

```html
<html>
  <head>
    <script src="/ProcessNet/html/Dialogs/scr/PNDialogs.js"></script>
  </head>
  <body>
    <button id=btnDialog>Click Me</button>
  </body>
</html>
```
PromptUser

The PromptUser dialog box function prompts the user for text, and returns the text typed by the user. The function requires the parameters shown in the following table.

Parameters for PromptUser Function

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sTitle</td>
<td>String</td>
<td>The title for the dialog box.</td>
</tr>
<tr>
<td>sPrompt</td>
<td>String</td>
<td>The prompt to be displayed to the user.</td>
</tr>
<tr>
<td>sDefault</td>
<td>String</td>
<td>The default text for the edit field.</td>
</tr>
<tr>
<td>sHelpURL</td>
<td>String, optional</td>
<td>The URL that contains help for the current prompt.</td>
</tr>
<tr>
<td>sIcon</td>
<td>String, optional</td>
<td>The URL for the icon you want to be displayed.</td>
</tr>
<tr>
<td>nMaxLength</td>
<td>Number, optional</td>
<td>The maximum length of the text.</td>
</tr>
<tr>
<td>funcValidation</td>
<td>Function pointer, optional</td>
<td>The function pointer you want to use to validate the input.</td>
</tr>
</tbody>
</table>

Example: Using PromptUser (PromptUser.html).

```html
<SCRIPT defer=true>
    function btnDialog.onclick()
    {
        var sText = PromptUser("Test Dialog", "Enter some text:", "");
        AlertDialog(sText);
    }
</SCRIPT>
</HEAD>
<BODY>
    <BUTTON id=btnDialog>Click Me</BUTTON>
</BODY>
</HTML>
```
ConfirmDialog

Like the AlertDialog function, ConfirmDialog uses the MessageDialog function to display a message with OK and Cancel buttons. ConfirmDialog returns true if OK is clicked, and false if Cancel is clicked.

Example: Using ConfirmDialog (ConfirmDialog.html).

```html
<HTML>
<HEAD>
<SCRIPT src="/ProcessNet/html/Dialogs/scr/PNDialogs.js">
</SCRIPT>
<SCRIPT defer=true>
function btnDialog.onclick()

    var sBtn = ConfirmDialog("This will work. Continue?");
    AlertDialog(sBtn);

</SCRIPT>
</HEAD>
<BODY>
<BUTTON id=btnDialog>Click Me</BUTTON>
</BODY>
</HTML>
```

ErrorMessage

The ErrorMessage dialog box function is similar to the AlertDialog function, except for the dialog box title and icon.

Example: Using ErrorMessage (ErrorMessage.html).

```html
<HTML>
<HEAD>
<SCRIPT src="/ProcessNet/html/Dialogs/scr/PNDialogs.js">
</SCRIPT>
<SCRIPT defer=true>
function btnDialog.onclick()

    ErrorMessage("Something bad happened");

</SCRIPT>
</HEAD>
<BODY>
<BUTTON id=btnDialog>Click Me</BUTTON>
</BODY>
</HTML>
```

ExtendedAlertDialog

You can use the ExtendedAlertDialog function to display a user-friendly error message, along with a more detailed and technical error message. The user can see the detailed error message by clicking Details on the dialog box. The ExtendedAlertDialog function takes the user-friendly and detailed messages as parameters.

Example: Using ExtendedAlert (ExtendedAlert.html).
Application Dialog Boxes

The following table describes some of the application dialog boxes available in Operational Insight. You can access all of these dialog boxes by including the PNDialogs.js file in your pages.

### Operational Insight Dialog Boxes

<table>
<thead>
<tr>
<th>Dialog Box Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Save</td>
<td>Specifies save file names, permissions, and tree link names for saved items.</td>
</tr>
<tr>
<td>Load</td>
<td>Specifies the content to load from the server.</td>
</tr>
<tr>
<td>Color</td>
<td>Selects a color from a color palette.</td>
</tr>
<tr>
<td>Font</td>
<td>Selects a font.</td>
</tr>
<tr>
<td>Tag Search</td>
<td>Retrieves a list of tag names.</td>
</tr>
<tr>
<td>SQL Editor</td>
<td>Composes a SQL statement.</td>
</tr>
<tr>
<td>DateTime</td>
<td>Selects a date from a calendar.</td>
</tr>
</tbody>
</table>

### Save

The Save dialog box is a common dialog box used by Operational Insight applications to save content to the server. This dialog box enables users to specify file names, permissions, and tree link names for saved items.

You can display the Save dialog box by calling the OpenSaveFileDialog function, which takes the parameters shown in the following table.
### Parameters for OpenSaveFileDialog Function

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sToolName</td>
<td>String</td>
<td>The name of the Operational Insight application for which you want to save the content. This must be a valid tool name, as defined in NetManage.</td>
</tr>
<tr>
<td>sData</td>
<td>String</td>
<td>The data you want to save to the Operational Insight content database.</td>
</tr>
<tr>
<td>sDialogTitle</td>
<td>String</td>
<td>The title for the Save dialog box. (Deprecated. Needs to be specified, acts as a placeholder, but is ignored.)</td>
</tr>
<tr>
<td>sDialogHeader</td>
<td>String</td>
<td>The page header text that is displayed above the file list. (Deprecated. Needs to be specified, acts as a placeholder, but is ignored.)</td>
</tr>
<tr>
<td>oWDS</td>
<td>JWebDataSet</td>
<td>The WebDataSet object used for communication with the server (see “JWebDataSet Object” on page 96). (Deprecated. Needs to be specified, acts as a placeholder, but is ignored.)</td>
</tr>
<tr>
<td>sFileName</td>
<td>String</td>
<td>(Optional) The default filename you want to display in the dialog box.</td>
</tr>
<tr>
<td>sFileDesc</td>
<td>String</td>
<td>(Optional) The default description you want to display in the dialog box.</td>
</tr>
<tr>
<td>oTreeFrame</td>
<td>Frame</td>
<td>(Optional) A reference to the frame/window containing the user’s profile tree. This value is only required if you want the user’s profile tree to update as soon the file is saved. If you do not provide this parameter, the user may still save to the tree, but changes may not appear until the next time the user reloads the tree.</td>
</tr>
</tbody>
</table>
The function returns an array with the results of the operation, as shown in the following table.

### Array Index for OpenSaveFileDialog Function

<table>
<thead>
<tr>
<th>Array Index</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The status value used to specify whether the action was successful. If there were no errors, this value is 1. It is 0 if errors occurred.</td>
</tr>
<tr>
<td>1</td>
<td>The database ID (ConfigID) of the saved file; contains the error message if errors occurred during the operation.</td>
</tr>
<tr>
<td>2</td>
<td>The name of the saved file.</td>
</tr>
<tr>
<td>3</td>
<td>The description of the saved file.</td>
</tr>
</tbody>
</table>

### Load

The **Load** dialog box is a common dialog box used by Operational Insight applications to load content from the server.

You can display the **Load** dialog box by calling `OpenLoadFileDialog` function, which takes the parameters shown in the following table.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sIcon</td>
<td>String</td>
<td>(Optional) The icon file name to be used when creating a link in the tree. The default is html.gif.</td>
</tr>
<tr>
<td>bNewWin</td>
<td>Boolean</td>
<td>(Optional) Sets the <strong>Open In New Window</strong> property for the link in the tree. This defaults to false.</td>
</tr>
<tr>
<td>sServer</td>
<td>String</td>
<td>(Optional) Specifies the name of the server to connect to.</td>
</tr>
<tr>
<td>sExtension</td>
<td>String</td>
<td>(Optional) The file name extension appended to the file name entered by the user.</td>
</tr>
<tr>
<td>iStartPathID</td>
<td>Number</td>
<td>(Optional) The path ID of the path you want to use as the root node for the Content Database tree.</td>
</tr>
</tbody>
</table>
Parameters for OpenLoadFileDialog Function

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sToolName</td>
<td>String</td>
<td>The name of the Operational Insight application for which you want to save the content. This must be a valid tool name, as defined in NetManage.</td>
</tr>
<tr>
<td>sDialogTitle</td>
<td>String</td>
<td>The title for the dialog box.</td>
</tr>
<tr>
<td>sDialogHeader</td>
<td>String</td>
<td>The page header text that is displayed above the file list.</td>
</tr>
<tr>
<td>oWDS</td>
<td>JWebDataSet</td>
<td>The WebDataSet object used for communication with the server (see “JWebDataSet Object” on page 96).</td>
</tr>
<tr>
<td>bNoload</td>
<td>Boolean</td>
<td>(Optional) If true, the selected file is not loaded, and the returned array will hold an empty string at index 6.</td>
</tr>
<tr>
<td>oWindow</td>
<td>Window</td>
<td>(Optional) If specified, the dialog will be modal to this window. Defaults to being modal to the window loaded with PNDialog.js.</td>
</tr>
<tr>
<td>sServer</td>
<td>String</td>
<td>(Optional) Specifies the name of the server to which you want to connect.</td>
</tr>
</tbody>
</table>

The function returns an array with the results of the operation, as shown in the following table.

Array Index for OpenLoadFileDialog Function

<table>
<thead>
<tr>
<th>Array Index</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The database ID (ConfigID) for the content.</td>
</tr>
<tr>
<td>1</td>
<td>The name of the content that was loaded</td>
</tr>
<tr>
<td>2</td>
<td>The description of the content.</td>
</tr>
<tr>
<td>3</td>
<td>The name of the tool to which the content belongs.</td>
</tr>
<tr>
<td>4</td>
<td>The name of the author of the content.</td>
</tr>
<tr>
<td>5</td>
<td>The date the content was last updated.</td>
</tr>
<tr>
<td>6</td>
<td>The actual content loaded from the server.</td>
</tr>
</tbody>
</table>

The function returns an array with the results of the operation, as shown in the following table.
Color

The Color dialog box is a common dialog box used by Operational Insight applications to enable users to select a color. Color is selected by clicking a color in the color palette displayed in the dialog box.

You can display the Color dialog box by calling the launchColorWindow function. The function returns null if the user clicks Cancel. Otherwise, the function returns the color as a hexadecimal string.

Parameters for launchColorWindow Function

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sHex</td>
<td>String</td>
<td>(Optional) The initial color value in hexadecimal. Defaults to #000000.</td>
</tr>
<tr>
<td>oWindow</td>
<td>Window</td>
<td>(Optional) If specified, the dialog will be modal to this window. Defaults to being modal to the window loaded with PNDialog.js.</td>
</tr>
<tr>
<td>iPaletteNumber</td>
<td>Number</td>
<td>(Optional) The color palette you want to display.</td>
</tr>
</tbody>
</table>

Example: Using the Color dialog box (Color.html).

```html
<HTML>
  <HEAD>
    <SCRIPT src="/ProcessNet/html/Dialogs/scr/PNDialogs.js"></SCRIPT>
    <SCRIPT defer=true>
      function btnDialog.onclick()
      {
        var sColor = launchColorWindow("#000000");
        if(sColor!=null)
        {
          document.body.style.backgroundColor = sColor;
        }//if
      }
    </SCRIPT>
  </HEAD>
  <BODY>
    <BUTTON id=btnDialog>Click Me</BUTTON>
  </BODY>
</HTML>
```

Font

The Font dialog box is a common dialog box used by Operational Insight applications to enable users to select a font.

You can display the font dialog box by calling the getFont function, which takes the parameters shown in the following table.
Parameters for getFont Function

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sFontName</td>
<td>String</td>
<td>The name of the default font, such as Courier or Arial.</td>
</tr>
<tr>
<td>nFontSize</td>
<td>Integer</td>
<td>The default font size.</td>
</tr>
<tr>
<td>enumFontStyle</td>
<td>Enumeration</td>
<td>The font style. This can be one of the following values: ENUM_FONT_TYPE_PLAIN</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ENUM_FONT_TYPE_BOLD</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ENUM_FONT_TYPE_ITALIC</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ENUM_FONT_TYPE_BOLD_AND_ITALIC</td>
</tr>
</tbody>
</table>

The function returns the default font if the user clicks `Cancel`. Otherwise, the function returns an array with the font characteristics, as shown in the following table.

Array Index for getFont Function

<table>
<thead>
<tr>
<th>Array Index</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Font name.</td>
</tr>
<tr>
<td>1</td>
<td>Font size.</td>
</tr>
<tr>
<td>2</td>
<td>Font style (see the <code>enumFontStyle</code> parameter).</td>
</tr>
</tbody>
</table>

Tag Search

The `Tag Search` dialog box is a common dialog box used by Operational Insight applications to enable users to retrieve a list of tag names.

You can display the tag search dialog box by calling the `OpenTagListWindow` function, which takes the parameters shown in the following table.
Parameters for OpenTagListWindow Function

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sDS</td>
<td>String</td>
<td>The default data source name.</td>
</tr>
<tr>
<td>nValues</td>
<td>Integer</td>
<td>The number of tags that the user is allowed to select.</td>
</tr>
<tr>
<td>bNoTrend</td>
<td>Boolean</td>
<td>Specifies whether the dialog box is being used with the trend.</td>
</tr>
<tr>
<td>oWindow</td>
<td>Window object</td>
<td>(Optional; can be null) The window object that is the parent of the dialog box.</td>
</tr>
<tr>
<td>oWDS</td>
<td>JWebDataSet</td>
<td>The WebDataSet object used for communication with the server (see “JWebDataSet Object” on page 96).</td>
</tr>
<tr>
<td>oTrend</td>
<td>Trend object</td>
<td>(Optional) The trend you want to use for validation and other information.</td>
</tr>
<tr>
<td>bSQL</td>
<td>Boolean</td>
<td>Specifies whether to hide the Add SQL button.</td>
</tr>
<tr>
<td>sPath</td>
<td>String</td>
<td>(Optional) Specifies the server path from which you want the dialog box loaded.</td>
</tr>
<tr>
<td>sUserName</td>
<td>String</td>
<td>(Optional) Specifies the user name.</td>
</tr>
<tr>
<td>sPassword</td>
<td>String</td>
<td>(Optional) Specifies the password.</td>
</tr>
<tr>
<td>bSQLOnly</td>
<td>Boolean</td>
<td>(Optional) If true, only SQL-type data sources will be listed.</td>
</tr>
<tr>
<td>bNoColumn Restriction</td>
<td>Boolean</td>
<td>(Optional) If true, the three column restriction on SQL statements is disabled. This is useful in combination with bSQLOnly for getting generic SQL statements from the user.</td>
</tr>
</tbody>
</table>

The function returns an array with the results of the operation, as shown in the following table.
Array Index for OpenTagListWindow Function

<table>
<thead>
<tr>
<th>Array Index</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>An array of tag information which contains the name, description, and data source for each selected tag. For example, if the return value was assigned to the aRet variable: aRet[0][0] = first tag name aRet[0][1] = first tag descriptor aRet[0][2] = first tag data source aRet[1][0] = second tag name</td>
</tr>
<tr>
<td>1</td>
<td>The start time string; only if bNoTrend was false.</td>
</tr>
<tr>
<td>2</td>
<td>The end time string; only if bNoTrend was false.</td>
</tr>
<tr>
<td>3</td>
<td>The refresh interval string; only if bNoTrend was false.</td>
</tr>
<tr>
<td>4</td>
<td>The X-Axis name; only if bNoTrend was false.</td>
</tr>
<tr>
<td>5</td>
<td>The Y-Axis name; only if bNoTrend was false.</td>
</tr>
<tr>
<td>6</td>
<td>A Boolean value, indicating whether the Real-time only check box was selected.</td>
</tr>
<tr>
<td>7</td>
<td>The series type, only if bNoTrend was false. Can be Line, Bar, or Constant.</td>
</tr>
</tbody>
</table>

Example: Using the Tag Search dialog box (TagSearch.html).

```html
<HTML>
<HEAD>
<TITLE>OpenTagListWindow Example</TITLE>
<SCRIPT src="/ProcessNet/HTML/dialogs/scr/Pndialogs.js">
</SCRIPT>
<SCRIPT src="/ProcessNet/HTML/scr/SessionManager.js">
</SCRIPT>
<SCRIPT defer=true>
function btnClick.onclick() {
    // ensure a session exists
    var oSession = new SessionManager(wdsData);
    if (!oSession.confirmSession())
        // return if user doesn't login
        return;
} // if

    // create the variables to use as parameters for the
    // OpenTagListWindow function call
    var sDatasource = "OPC_SIM"; // Default database displayed
    var nValue = 10; // Max tags that can be selected
    var bNoTrend = true; // Hide the Trend specific inputs
    var oWin = window; // Calling window
    var oWDS = wdsData; // WebDataSet to use in queries
    var oTrend = null; // Trend to get data from
    var bSQL = true; // Show the "Add SQL" button
```
Application Dialog Boxes

```javascript
var sServer = ""; // Name of the server to log into
//(for EnterpriseNet)
var sUserName = ""; // UserName to log in with
//(for EnterpriseNet)
var sPassword = ""; // Password to log in with
//(for EnterpriseNet)
var bSQLOnly = false; // Show all data sources, not just
// SQL ones
var bNoColumnRestriction = false; // Don’t force queries to
// be tag compatible (3 columns)

// make the OpenTagListWindow function call and assign the
// return value to a variant
var vReturn = OpenTagListWindow(sDatasource, nValue,
  bNoTrend, oWin, oWDS, oTrend, bSQL, sServer, sUserName,
  sPassword, bSQLOnly, bNoColumnRestriction);
// check to see if the dialog box was cancelled
if (vReturn == null) // no data returned, so alert user
  alert("No data returned");
else
{
  // some data was returned, so find out how many results
  var vArrayElements = vReturn[0].length;
  // create a string to display the number of tags returned
  var vString = "There are " + vArrayElements + ", " +
    " tags returned from the OpenTagListWindow.";
  alert(vString);
} //if
</SCRIPT>
</HEAD>
<BODY>
<BUTTON ID= btnClick>
  Click Me
</BUTTON>
<APPLET
  CODE="JWebDataSet.JWebDataSet.class"
  ID="wdsData"
  style="Position:absolute; Top:1px; Left:1px; height:1; width:1;"
  archive="/ProcessNet/HTML/PNJTrend/PNJTrend.jar"
  MAYSRIPT = "true">
</APPLET>
<DIV ID=dvTable></DIV>
</BODY>
</HTML>

SQL Editor

The SQL Editor dialog box is a common dialog box used by Operational Insight applications to enable users to enter SQL.

You can display the SQL Editor dialog box by calling the OpenSQLInputDialog function, which takes the parameters shown in the following table.
Parameters for OpenSQLInputDialog Function

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sDS</td>
<td>String</td>
<td>The default data source name.</td>
</tr>
<tr>
<td>oWindow</td>
<td>Window object</td>
<td>(Optional; can be null) The window object that is the parent of the dialog box.</td>
</tr>
<tr>
<td>oWDS</td>
<td>JWebDataSet</td>
<td>The WebDataSet object used for communication with the server (see “JWebDataSet Object” on page 96).</td>
</tr>
<tr>
<td>sSQL</td>
<td>String</td>
<td>The default SQL you want to display.</td>
</tr>
</tbody>
</table>

The SQL Editor function returns an array with two elements, as shown in the following table.

Array Index for OpenSQLInputDialog Function

<table>
<thead>
<tr>
<th>Array Index</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The selected data source name.</td>
</tr>
<tr>
<td>1</td>
<td>The SQL entered by the user.</td>
</tr>
</tbody>
</table>

Tip
Using the Tag Search dialog in SQLOnly mode, with bNoColumnRestriction set to true might be a better choice than using OpenSQLInputDialog, because it provides a richer environment (history feature, integration with NetQuery).

FMI
For more information about using the Tag Search dialog box, see “Tag Search” on page 84.

DateTime
You can display the date and time selection dialog box by using the getDateTimeFromCalendar function. This function takes no parameters, and returns the user-selected date and time. Alternatively, the OpenCalendarDialog function can be used: it also takes no parameters and returns the user-selected date and time. Example: Using getDateTimeFromCalendar(Date-Time.html).
<SCRIPT src="/ProcessNet/html/Dialogs/scr/PNDialogs.js">
</SCRIPT>
<SCRIPT defer=true>
function btnDialog.onclick(){
   var sTime = getDateTimeFromCalendar();
   AlertDialog(sTime);
}
</SCRIPT>
</HEAD>
<BODY>
   <BUTTON id=btnDialog>Click Me</BUTTON>
</BODY>
</HTML>
Chapter 7: Operational Insight Global Functions

Operational Insight uses a number of functions that you can use in your own applications. The following sections describe the global functions available.

Numerical

You can use several functions to manipulate numerical values. These functions are available in the numbers.js JavaScript file, and are shown in the following table.

Numerical Functions in numbers.js File

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>intToHex</td>
<td>Converts an integer value to a hexadecimal string.</td>
</tr>
<tr>
<td>hexToInt</td>
<td>Converts a hexadecimal string to an integer value.</td>
</tr>
<tr>
<td>FormatFloat</td>
<td>Formats a floating point number with the specified number of decimal places.</td>
</tr>
</tbody>
</table>

Example: Using the numerical functions (NumericalFunctions.html).

```html
<html>
<head>
<script src="/ProcessNet/html/scr/numbers.js"></script>
<script defer>
function btnToInt.onclick()
{
  alert( hexToInt(edtHex.value) );
}

function btnToHex.onclick()
{
  alert( intToHex(edtInt.value) );
}

function btnFormat.onclick()
{
  alert( FormatFloat(edtFloat.value, "6.2") );
}
</script>
</head>
<body>
</body>
```
Date/Time

You can use several functions to manipulate date and time values. These functions are available in the `time.js` JavaScript file, and are shown in the following table.

Date and Time Functions in `time.js` File

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getDateTime</td>
<td>Returns the current date as a string formatted as MM/DD/YYYY hh:mm.</td>
</tr>
<tr>
<td>dateToMinutes</td>
<td>Converts the supplied date to the number of minutes since January 1, 1970.</td>
</tr>
<tr>
<td>minutesToDate</td>
<td>Converts the supplied number of minutes to the date, starting at January 1, 1970.</td>
</tr>
<tr>
<td>dateToSeconds</td>
<td>Converts the supplied date to the number of seconds since January 1, 1970.</td>
</tr>
<tr>
<td>FormatDateString</td>
<td>Specifies the format for the JavaScript Date object.</td>
</tr>
</tbody>
</table>

Example: Using the time functions (`TimeFunctions.html`).

```html
<html>
<head>
  <script src="/ProcessNet/html/scr/time.js"></script>
  <script defer>
    //btnGetDate.onclick()
    function btnGetDate.onclick()
    {
      alert( getDateTime() );
    }

    //btnToMinutes.onclick()
    function btnToMinutes.onclick()
    {
      alert( dateToMinutes( getDateTime() ) );
    }

    //btnToSeconds.onclick()
    function btnToSeconds.onclick()
    {
      alert( dateToSeconds( getDateTime()+"":00" ) );
    }
  </script>
</head>
<body>
</body>
</html>
```
Global Variables

Global variables and categories are defined in NetManage by using the Global Variable link. All of the variables that are defined here are available through PNSP, but must be made available through script by editing the file:
\ProcessNet\Web<html\scr\GlobalVars.pnsp

You can get access to global variables through script by including the GlobalVars.pnsp file in a document. The file contains JavaScript methods for obtaining the values defined for global variables.

Global Variable Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getGlobalValue</td>
<td>Takes an argument for the name of the category and an argument for the name of the global variable. Returns the value of the global variable from Operational Insight.</td>
</tr>
</tbody>
</table>

Example: Using a global variable in a script.

```html
<HTML>
<HEAD>
<SCRIPT src = "/ProcessNet/html/scr/GlobalVars.pnsp"
</SCRIPT>
<SCRIPT>
function ShowDateFormat()
{
    vars sFormat = getGlobalValue("General", "DateFormat");
    DateFormatCell.innerText = sFormat;
}
</SCRIPT>
</HEAD>
<BODY onload = "ShowDateFormat()">
```
| DateFormat: | }
|---|---

```html
<TABLE border = 1>
  <TR>
    <TD>DateFormat:</TD>
    <TD id = DateFormatCell>&nbsp;</TD>
  </TR>
</TABLE>
```
Chapter 8: Using the WebDataSet

You can use the WebDataSet to get access to Operational Insight data from Web pages. The WebDataSet is a Java applet that scripts can use to retrieve all types of data from the Operational Insight server. The command syntax for the WebDataSet is the same as that used for URL queries.

Including the Applet

To use the WebDataSet, you must include an <APPLET> element in the HTML page:

```html
<APPLET id = PNWebDS
code = JWebDataSet.JWebDataSet
archive = "/ProcessNet/HTML/PNJTrend/PNJTrend.jar"
width = 0
height = 1
MAYSCRIPT>
</APPLET>
```

Applet Parameters

The applet can also use parameters that affect the applet's behavior. The following table describes these parameters.

### Applet Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>QueryPath</td>
<td>The path to the NetPortal.</td>
<td>/ProcessNet/isapi/netportal/netportal.dll</td>
<td></td>
</tr>
<tr>
<td>OnLogin</td>
<td>Sets the name of the JavaScript function you want to call when the user is required to log on. This is normally called if the session expires.</td>
<td>OnLogin</td>
<td></td>
</tr>
<tr>
<td>BgColor</td>
<td>Sets the background color for the applet.</td>
<td>Gray</td>
<td></td>
</tr>
</tbody>
</table>

If the `OnLogin` parameter is not specified, and an `openRecordSet` call is made while the user does not have a session, the page that displays the applet will be redirected to the Operational Insight logon page.

Because of the way in which Java threads work, it is safest to avoid processing in the event handler. Instead, use a `setTimeout` call with a short delay, and handle
the processing in another function. Otherwise, the browser may fail to respond in certain instances, especially if the event handler accesses an applet.

WebDataSet Objects

The WebDataSet applet provides access to five main objects:

• JWebDataSet: Provides methods for getting access to the other objects.
• JWebRecordSet: Holds all data returned from a request. Accessed through the JWebDataSet.
• XField: Provides access to individual columns in the returned data. Accessed through the JWebDataSet.
• MetaDataSet: Used to get access to metadata from the server.
• PNDataSourceList: Returns the type of a specified data source (0 for tag, 1 for SQL, and 2 for SQL to Tag).

JWebDataSet Object

The JWebDataSet is the main object in the applet. Use the JWebDataSet to get access to the other objects, such as the JWebRecordSet (or record set) object. Use the openRecordset method with the WebDataSet object to execute a command against the server and return a record set object (JWebRecordSet) that contains the data returned from the server.

The openRecordset method takes the parameters shown in the following table.

Parameters for openRecordSet Function in JWebDataSet

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| sPath     | String     | The path to the NetPortal and data source. For example: /ProcessNet/isapi/netportal/netportal.dll/data/[ds]
Where [ds] is the name of the data source to which you want to get access. |
| sCommand  | String     | The command you want to pass to the NetPortal. This command follows the same syntax as URL queries. |

The openRecordset call is a blocking call, which means that the function call does not return until the query results have been retrieved. For large queries, this can cause the browser to appear unresponsive for several seconds. The following example shows how to open a non-blocking query.

```javascript
var reader=PNWebDS.createReader("GotResults");
reader.openRecordset("OPC_SIM","Tag0=Random.Int1");
alert("Query Start");
function GotResults(rsResults, sErrorString, appWebDS, sURL, sCommand){
    alert("Query Completed");
```
The `openRecordset` method returns a record set object.

Example: Sending a command to the server (Command.html).

```html
<HTML>
<HEAD>
<script defer=true>
  var sPath="/ProcessNet/isapi/netportal/netportal.dll/data/";
  function btnData.onclick()
  {
    var rsResults = PNWebDS.openRecordset(sPath+"OPC_SIM",
      "tag0=Random.Int2");
  }
</SCRIPT>
</HEAD>
<BODY>
  <BUTTON id=btnData>Get Data</BUTTON>
  <APPLET id=PNWebDS
    code=JWebDataSet.JWebDataSet
    archive="/ProcessNet/HTML/PNJTrend/PNJTrend.jar"
    width=0
    height=1
    MAYSCTRIPT>
  </APPLET>
</BODY>
</HTML>
```

Note

Unlike URL queries, you cannot specify a format for the `WebDataSet` commands.

The tags within the command must be passed with the JavaScript `escape` function if they contain non-alphanumeric characters. The SQL statements are automatically parsed without the need for this type of intervention, provided that `sql=` prefixes the statement.

Example: Using `escape` in a `WebDataSet` request (Request.html).

```html
<HTML>
<HEAD>
<script defer=true>
  var sPath="/ProcessNet/isapi/netportal/netportal.dll/data/";
  var sTag1 = escape( "Random.Int2" );
  var sTag2 = escape( "Random.Int1" );
  function btnData.onclick()
  {
    var rsResults = PNWebDS.openRecordset(sPath+"OPC_SIM",
      "tag0=" + sTag1 + "+tag1=" + sTag2);
  }
</SCRIPT>
</HEAD>
<BODY>
  <BUTTON id=btnData>Get Data</BUTTON>
  <APPLET id=PNWebDS
    code=JWebDataSet.JWebDataSet
    archive="/ProcessNet/HTML/PNJTrend/PNJTrend.jar"
    width=0
    height=1
    MAYSCTRIPT>
  </APPLET>
</BODY>
</HTML>
```
**JWebRecordSet Object**

The `openRecordset` method of the WebDataSet returns a record set object that contains the results of the request. These results may be either an error or data from the request, and can be obtained by using the record set methods. The most common methods used with the record set are described in the following table.

**JWebRecordSet Functions**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetLastError</td>
<td>Returns a string representing any errors that occurred while opening the record set, or returns NULL if no errors occurred.</td>
</tr>
<tr>
<td>First</td>
<td>Moves the record pointer to the first record returned.</td>
</tr>
<tr>
<td>Next</td>
<td>Moves the record pointer to the next record.</td>
</tr>
<tr>
<td>Last</td>
<td>Moves the record pointer to the last record returned.</td>
</tr>
<tr>
<td>Prior</td>
<td>Moves the record pointer to the previous record.</td>
</tr>
<tr>
<td>Eof</td>
<td>Returns true if the record pointer is at the end of the results.</td>
</tr>
<tr>
<td>Fields</td>
<td>Returns a reference to a specific field in the result set by numerical index.</td>
</tr>
<tr>
<td>FieldByName</td>
<td>Returns a reference to a specific field in the result set by field name.</td>
</tr>
<tr>
<td>FieldCount</td>
<td>Returns the number of fields in the result set.</td>
</tr>
<tr>
<td>RecordCount</td>
<td>Returns the number of rows in the result set.</td>
</tr>
</tbody>
</table>

**FMI**

For more information, and an example of using the record set, see “XField Object” on page 98.

**XField Object**

The `Recordset` object represents a table of values. The record pointer in the record set is used to index specific rows within the table (using `Next`, `Prior`, and `EoF` methods). The `XField` object gives you access to the individual columns within the data table stored in the record set object.
The XField object provides the methods shown in the following table.

**XField Functions**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getType</td>
<td>Returns the data type for the field, as an integer. For a description of the integer values, see “Values for the odbcDataType Parameter” on page 51.</td>
</tr>
<tr>
<td>getName</td>
<td>Returns the name of the field.</td>
</tr>
<tr>
<td>isNull</td>
<td>Returns true if the field is null; always returns false unless WantNulls = true or WantNulls = 1 is included in the data request command.</td>
</tr>
<tr>
<td>AsString</td>
<td>Formats the field data as a string.</td>
</tr>
<tr>
<td>AsInteger</td>
<td>Formats the field data as an eight-byte integer.</td>
</tr>
<tr>
<td>AsFloat</td>
<td>Formats the field data as an eight-byte floating point value.</td>
</tr>
<tr>
<td>AsDate</td>
<td>Formats the field data as a Java Date object.</td>
</tr>
<tr>
<td>AsTimeString</td>
<td>Formats the field data as a date and time string.</td>
</tr>
</tbody>
</table>

Since the XField function, isNull, returns true if the field is null, it will always return false unless WantNulls = true or WantNulls = 1 is included in the data request command. The following example illustrates isNull:

```javascript
var oWebDataSet = RID_LoadWebDS;
var sQuery = "select name, sizecol from animals";
var sRequest = "WantNulls=true&sql= " + escape(sQuery);
var sDatasource = "SQL_DS";
var oRS = oWebDataSet.openRecordSet(sDatasource, sRequest);
if (oRS != null) {
    if (oRS.GetLastError() != "") {
        alert("Error: " + oRS.GetLastError());
    }
    var oFieldName = oRS.Fields(0);
    var oFieldSize = oRS.Fields(1);
    oRS.First();
    while (!oRS.Eof()) {
        if (oFieldSize.isNull()) {
            alert("The field value is null for " + oFieldName.asString());
        } else {
            alert("Name: " + oFieldName.asString() + ", size: " + oFieldSize.asString());
        }
        oRS.Next()
    }
}
```

Avoid repeating function or method calls unnecessarily, such as when a function call is executed repeatedly in a loop, especially when the result could be stored in
Chapter 8: Using the WebDataSet

a local variable. In the following example, calling the FieldByName method returns a reference to an XField object, which remains valid as long as the result set is open. FieldByName need only be called once for each field that must be accessed. This is why it is outside the while loop.

Example: Displaying results from a WebDataSet query (Results.html).

```html
<HTML>
<HEAD>
<script defer=true>
var sPath = "/ProcessNet/isapi/netportal/netportal.dll/data/";
var sSQL = "select Company, State from customer";

function btnData.onclick()
{
  var rsResults = PNWebDS.openRecordset(sPath + "SQL_DS", "sql=\" + sSQL);
  if (rsResults.GetLastError() != "")
  {
    alert("Error: \" + rsResults.GetLastError());
    return;
  } //if
  var fldName = rsResults.FieldByName("Company");
  var fldState = rsResults.FieldByName("State");
  var sList = "";
  rsResults.First();
  while(rsResults.Eof() == false)
  {
    sList += fldName.AsString() + ",";
    sList += fldState.AsString() + "<BR>";
    rsResults.Next();
  } //while
  dvTable.innerHTML = sList;
}
</script>
</HEAD>
<BODY>
<button id=btnData>Get Data</button>
<applet id=PNWebDS
  code=JWebDataSet.JWebDataSet
  archive="/ProcessNet/HTML/PNJTrend/PNJTrend.jar"
  width=0
  height=1
  MAYSRCSCRIPT>
</applet>
<!-- container for the table -->
<div id=dvTable></div>
</BODY>
</HTML>

Accessing Metadata

You can gain access to metadata using the WebDataSet and the queries described in the URL query section. However, the WebDataSet provides wrapper classes for metadata that make getting access to it easier.

The main object used to get access to metadata is the MetaDataSet object, which can be retrieved from the JWebDataSet using the metaData method. The MetaDataSet provides several functions that map directly to the metadata requests supported by the NetPortal (see “Basic Metadata Requests” on page 38).
Several helper objects are also available to provide enumerations for the metadata attributes that can be accessed, as shown in the following table.

### Helper Objects for Gaining Access to Metadata Attributes

<table>
<thead>
<tr>
<th>Object</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MetaDataTypes</td>
<td>Enables you to enumerate through the metadata types supported by the server.</td>
</tr>
<tr>
<td>TagAttributes</td>
<td>Defines named variables for each tag attribute that can be requested.</td>
</tr>
<tr>
<td>TagInterfaces</td>
<td>Defines named variables for each type of tag interface supported by the server: real-time, history, or auto.</td>
</tr>
</tbody>
</table>

Example: Building a list box with the list of all possible tag attributes (MetadataListbox1.html).

```html
<HTML>
<HEAD>
<SCRIPT>
function doOnLoad()
{
    var mdtTypes = PNWebDS.newObject("JWebDataSet.MetaDataTypes");
    var eTagAttr = mdtTypes.enumTagAttributes();
    var oAttr;
    while( eTagAttr.hasMoreElements() )
    {
        oAttr = eTagAttr.nextElement();
        var oOption = document.createElement("OPTION");
        oOption.text = oAttr.getName();
        oOption.value = oAttr.getValue();
        cbAttr.add(oOption);
    }//while
}
</SCRIPT>
</HEAD>
<BODY onload="doOnLoad()"
    <SELECT id=cbAttr>
        <APPLET id=PNWebDS
            code=JWebDataSet.JWebDataSet
            archive="/ProcessNet/HTML/PNJTrend/PNJTrend.jar"
            width = 0
            height = 1
            MAYSRIPT>
    </APPLET>
</BODY>
</HTML>
```

Example: Accessing tag attributes using the enumeration defined in TagAttributes (MetadataListbox2.html).

```html
<HTML>
<HEAD>
<SCRIPT defer>
function btnData.onclick()
{
```

---

*Operational Insight R362.1 Programmer’s Guide Page 101*
Chapter 8: Using the WebDataSet

```javascript
var mdsMetaSet = PNWebDS.metaData();
var mdtAttr = PNWebDS.newObject("JWebDataSet.TagAttributes");
var mdtItfc = PNWebDS.newObject("JWebDataSet.TagInterfaces");
PNWebDS.DatabaseName = "/ProcessNet/isapi/netportal/netportal.dll/data/opc_sim";
var rsResults = mdsMetaSet.tagAttribute("random.int2", mdtAttr.DESCRIPTION, mdtItfc.AUTO);
if (rsResults.GetLastError() != "") {
    alert("Error: "+ rsResults.GetLastError());
    return;
} //if
dvInfo.innerText = rsResults.Fields(1).AsString();
</SCRIPT>
</HTML>

Example: Determining the type of a data source. Only supports types tag (0), SQL (1), and SQL to tag (2) (no example file).

```javascript
var dstypes = PNWebDS.newObject("JWebDataSet.PNDataSourceList");
alert(dstypes.getDSType("OPC_SIM"));
```

**FMI**

The preceding sections describe only portions of the programming interface. For more information, refer to the JavaDocs, located in the `javadocs` folder on the installation CD.
Chapter 9: NetTrend Programming Interface

The Operational Insight trend object, NetTrend, provides an extensive programming interface that you can use to create custom charting and trending applications.

Including the Applet

To use the trend, you must include an `<APPLET>` element in the HTML page:

```html
<APPLET id = PNJTrend
   code = PNJTrend.BasicTrend
   archive = "/ProcessNet/HTML/PNJTrend/PNJTrend.jar"
   width = 100%
   height = 100%
   MAYSERIPT>
</APPLET>
```

Applet Parameters

The applet can also use a number of parameters that affect the trend’s behavior. The following table describes the parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>Sets the trend’s ID. This ID can be retrieved from the trend.</td>
<td>none</td>
</tr>
<tr>
<td>configID</td>
<td>The configID of a trend you want to immediately load.</td>
<td>none</td>
</tr>
<tr>
<td>QueryPath</td>
<td>The path to the NetPortal.</td>
<td>/ProcessNet/isapi/netportal/netportal.dll</td>
</tr>
<tr>
<td>RealTimeRetryCount</td>
<td>The number of consecutive errors you want to be ignored during real-time data access.</td>
<td>3</td>
</tr>
</tbody>
</table>
## Parameter Description Default

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>OnDblClick</td>
<td>Sets the name of the JavaScript function you want to call when the user double-clicks the trend.</td>
<td>TrendDblClick</td>
</tr>
<tr>
<td>OnClick</td>
<td>Sets the name of the JavaScript function to call when the user clicks the trend.</td>
<td>none</td>
</tr>
<tr>
<td>OnError</td>
<td>Sets the name of the JavaScript function you want to call when an error occurs in the trend.</td>
<td>none</td>
</tr>
<tr>
<td>DSTOffset</td>
<td>Sets the offset to be used during Daylight Saving Time in ms. (Deprecated. No replacement parameter required.)</td>
<td>0</td>
</tr>
<tr>
<td>HideCursorButton</td>
<td>When true, prevents the display of the cursor button in the top-left of the trend.</td>
<td>false</td>
</tr>
<tr>
<td>Title</td>
<td>The title of the trend.</td>
<td>none</td>
</tr>
<tr>
<td>DrawFrame</td>
<td>When false, prevents the drawing of the 3-D border around the trend.</td>
<td>true</td>
</tr>
<tr>
<td>AllowRightClickUnzoom</td>
<td>When false, disables un-zooming by right-click.</td>
<td>true</td>
</tr>
<tr>
<td>XML</td>
<td>The trend XML you want to immediately load.</td>
<td>none</td>
</tr>
<tr>
<td>DateFormat</td>
<td>The default display format for the trend axes.</td>
<td>M/d h:mm</td>
</tr>
</tbody>
</table>
Caution

Because of the way in which Java threads work, it is safest to avoid processing in the event handler. Instead, use a `setTimeout` call with a short delay, and handle the processing in another function. Otherwise, the browser may fail to respond in certain instances, especially if the event handler accesses an applet.

### Configuring Trends Programmatically

The trend’s programming interface can be used to add series, add axes, and configure all elements of the trend’s display. The trend exposes a number of objects that can be accessed from JavaScript to configure the trend.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParseDateFormat</td>
<td>Determines the formatting of start and end parameters in date requests. This is typically not used in the actual applet definition, but set afterwards through JavaScript. It must be set to the Operational Insight global variable for DateFormat, but in Java date format syntax (see time.js for conversion utilities).</td>
<td>DateFormat global variable default value</td>
</tr>
<tr>
<td>AllowAxesMove</td>
<td>When false, the axes cannot be resized or moved.</td>
<td>true</td>
</tr>
<tr>
<td>Debug</td>
<td>When true, additional logging is output to the console.</td>
<td>false</td>
</tr>
</tbody>
</table>
Chapter 9: NetTrend Programming Interface

Trend Objects

<table>
<thead>
<tr>
<th>Object</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BasicTrend</td>
<td>The applet object. This object exposes a few methods to interact with JavaScript, but is really just the container for the chart object.</td>
</tr>
<tr>
<td>PNJChart</td>
<td>The main chart object. This object provides access to the series list, axis list, cursor list, chart title, and chart legend.</td>
</tr>
<tr>
<td>PNChartLegend</td>
<td>The chart legend.</td>
</tr>
<tr>
<td>PNChartTitle</td>
<td>The chart title.</td>
</tr>
<tr>
<td>PNSeriesList</td>
<td>A list containing all series objects on the trend.</td>
</tr>
<tr>
<td>AxesList</td>
<td>A list containing all axes objects on the trend.</td>
</tr>
<tr>
<td>CursorList</td>
<td>A list containing all cursor objects on the trend.</td>
</tr>
<tr>
<td>PNChartSeries</td>
<td>The base series object. A number of series types derive from PNChartSeries, including bar series, constant series, and function series.</td>
</tr>
<tr>
<td>PNChartAxis</td>
<td>The base axis object. The horizontal and vertical axes derive from this base object.</td>
</tr>
<tr>
<td>PNDatasourceManager</td>
<td>Controls the data access for the series.</td>
</tr>
</tbody>
</table>

The following sections describe basic procedures to be used when manipulating the trend.

Adding a Line Series

Adding a series is the most common task when programming with the trend. You can add a series that automatically retrieves data from Operational Insight, or add a series to which you can manually add data.

To add line series to the trend, use one of the addSeries methods provided by the chart object. There are two overloaded addSeries methods: one enables you to manually add data, and the other retrieves data from the server. The addSeries method that retrieves data from the server uses the parameters shown in the following table.
Parameters for addSeries Function for Retrieving Data

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sTitle</td>
<td>String</td>
<td>The title for the series in the legend.</td>
</tr>
<tr>
<td>HistSrc</td>
<td>String</td>
<td>The data source for history data. This must be a fully qualified path to</td>
</tr>
<tr>
<td>HistTag</td>
<td>String</td>
<td>The tag for which you want to retrieve history data.</td>
</tr>
<tr>
<td>Start</td>
<td>String</td>
<td>The start time for history data.</td>
</tr>
<tr>
<td>End</td>
<td>String</td>
<td>The end time for history data.</td>
</tr>
<tr>
<td>Step</td>
<td>String</td>
<td>The step (interval) for data in the history request.</td>
</tr>
<tr>
<td>Function</td>
<td>String</td>
<td>The aggregate function to use when getting access to data, such as interp,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>comp, or plot.</td>
</tr>
<tr>
<td>RTSrc</td>
<td>String</td>
<td>The real-time data source name. This must be a fully qualified path to</td>
</tr>
<tr>
<td>RTTag</td>
<td>String</td>
<td>The tag from which you want to retrieve real-time data.</td>
</tr>
<tr>
<td>Refresh</td>
<td>Integer</td>
<td>The interval at which you want to retrieve real-time data in seconds.</td>
</tr>
</tbody>
</table>

The addSeries method that creates an empty series uses the parameters shown in the following table.

Parameters for addSeries Function for Creating an Empty Series

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sTitle</td>
<td>String</td>
<td>The title for the series in the legend.</td>
</tr>
<tr>
<td>Active</td>
<td>Boolean</td>
<td>Indicates whether the series is initially enabled or not.</td>
</tr>
</tbody>
</table>

Both versions of the addSeries method return a PNChartSeries object.

PNChartSeries Object

The PNChartSeries object represents a simple line series. All other series types (bar, constant, function) are derived from PNChartSeries. The PNChartSeries object provides a number of methods for manipulating the series.
The most common method used with a series is `AddXYStatus`, which adds a value to the series. The trend provides several variations of this method:

```java
public void AddXYStatus(double X, double Y, int nStatus)
public void AddXYStatus(double X, double Y, int nStatus, String sLabel)
public void AddXYStatus(double X, double Y, int nStatus, boolean bShowIfBad, Color clBadColor, String sLabel)
public void AddXYStatus(double X, double Y, int nStatus, boolean bShowIfBad, Color clBadColor, String sLabel)
```

The parameters for this method are shown in the following table.

**Parameters for AddXYStatus Function**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>Double</td>
<td>The X value for the data point.</td>
</tr>
<tr>
<td>Y</td>
<td>Double</td>
<td>The Y value for the data point.</td>
</tr>
<tr>
<td>nStatus</td>
<td>Integer</td>
<td>The status value for the data point.</td>
</tr>
<tr>
<td>sLabel</td>
<td>String</td>
<td>The label to display when the cursor moves over the point.</td>
</tr>
<tr>
<td>bShowIfBad</td>
<td>Boolean</td>
<td>Indicates whether to draw a line between points if the status value is bad.</td>
</tr>
<tr>
<td>clBadColor</td>
<td>Color</td>
<td>The color you want to use to draw the line for a bad status.</td>
</tr>
</tbody>
</table>

Example: Adding a series and data (`AddStaticSeries1.html`).

```html
<HTML>
<HEAD>
<SCRIPT defer>
function btnSeries.onclick()
{
    var ctChart = PNJTrend.getChart();
    var csSeries = ctChart.addSeries("My Series", true);
    csSeries.addXYStatus(1, 1, 192);
    csSeries.addXYStatus(2, 2, 192, "Some Text");
    csSeries.addXYStatus(3, 3, 192);
    PNJTrend.repaint();
}
</SCRIPT>
</HEAD>
<BODY>
<APPLET id=PNJTrend
    code=PNJTrend.BasicTrend
    archive = "/ProcessNet/HTML/PNJTrend/PNJTrend.jar"
    width = 100%
    height = 80%
    MAYSCRIPT>
</APPLET>
<BUTTON id=btnSeries>Add Series</BUTTON>
</BODY>
</HTML>
```
The 192 value passed into AddXYStatus is the OPC status code for OPC_QUALITY_GOOD. The codes that Operational Insight supports are found in PNAnimationLib.js:

```javascript
OPC_QUALITY_GOOD = 192,
OPC_QUALITY_SUB_NORMAL = 88,
OPC_QUALITY_SENSOR_CAL = 80,
OPC_QUALITY_LAST_USABLE = 68,
OPC_QUALITY_UNCERTAIN = 64,
OPC_QUALITY_OUT_OF_SERVICE = 28,
OPC_QUALITY_COMM_FAILURE = 24,
OPC_QUALITY_LAST_KNOWN = 20,
OPC_QUALITY_SENSOR_FAILURE = 16,
OPC_QUALITY_DEVICE_FAILURE = 12,
OPC_QUALITY_NOT_CONNECTED = 8,
OPC_QUALITY_CONFIG_ERROR = 4,
OPC_QUALITY_BAD = 0,
PN_TAG_NOT_FOUND = -5
```

Example: Adding a series with data from a WebDataSet (AddStaticSeries2.html).

```html
<HTML>
<HEAD>
  <SCRIPT defer>
  function btnSeries.onclick() {
    var ctChart = PNJTrend.getChart();
    var csSeries = ctChart.addSeries("My Series", true);
    var sPath = "/ProcessNet/isapi/netportal/netportal.dll/data/";
    var rsData = PNWebDS.openRecordset(sPath+"sql_ds", "sql=select

calledate, itemstotal from orders where custno=1351");
    if(rsData.GetLastError() != ")
    { 
      alert(rsData.GetLastError());
      return;
    }
    var fldXVal = rsData.FieldByName("SaleDate");
    var fldYVal = rsData.FieldByName("ItemsTotal");
    while( rsData.Eof() == false )
    {
      csSeries.AddXYStatus( fldXVal.AsFloat(),
        fldYVal.AsFloat(), 192);
      rsData.Next();
    }
    PNJTrend.repaint();
  } //btSeries.onclick()
</SCRIPT>
</HEAD>
</BODY>
  <APPLET id=PNJTrend
    code=PNJTrend.BasicTrend
    archive = "*/ProcessNet/HTML/PNJtrend/PNJtrend.jar"
```
Chapter 9: NetTrend Programming Interface

PNDataSourceManager Object

The PNDataSourceManager object in the trend is responsible for retrieving data for the series in the trend. It consolidates requests to reduce the amount of communication between the trend and server.

When a data series is added to the trend, the request for the data is not made immediately. Instead, the request is stored by the data source manager, and is not actually executed until told to get the data for the trend.

You need to know about the data source manager because you must explicitly tell it to get data after you add data series. Otherwise, the trend will not display data.

Example: Adding data series (AddDynamicSeries.html).

```html
<HTML>
<HEAD>
<SCRIPT defer>
function btnSeries.onclick()
{
  var sPath = "/ProcessNet/isapi/netportal/netportal.dll/data/";
  var ctChart = PNJTrend.getChart();
  ctChart.addSeries("Random 1", sPath+"OPC_SIM",
  "Random.Int1", "*-10M", "*", "500", "PLOT",
  sPath+"OPC_SIM", "Random.Int1", 1);
  ctChart.addSeries("Random 2", sPath+"OPC_SIM",
  "Random.Int2", "*-10M", "*", "500", "PLOT",
  sPath+"OPC_SIM", "Random.Int2", 1);
  ctChart.getDSManager().getData();
}
</SCRIPT>
</HEAD>
<BODY>
<APPLET id=PNJTrend
  code=PNJTrend.BasicTrend
  archive="/ProcessNet/HTML/PNJtrend/PNJtrend.jar"
  width=100%
  height=80%
  MAYSCRIPT>
</APPLET>
<BUTTON id=btnSeries>Add Series</BUTTON>
</BODY>
</HTML>
```

Example: Modifying data access parameters on existing series (ModifySeries.html).

```html
<HTML>
<HEAD>
```
Adding Functions

The trend has a number of built-in functions that you can configure programmatically. Functions can add, subtract, multiply, or divide a series. They can also calculate a series’ average, moving average, and standard deviation.

In order to add a function to the trend, you must first add a function series. You must then configure the function series with the appropriate function and the source series on which you want to base the calculation.

To add a function series, use the `addFunctionSeries` method in the chart object. This method takes two parameters, the series title and the function name. The function name represents a function object that is to be used to do the calculation. The standard functions available are as shown in the following table.

### Standard Functions Available

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add</td>
<td>Adds two series together.</td>
</tr>
<tr>
<td>Subtract</td>
<td>Subtracts one series from another.</td>
</tr>
<tr>
<td>Divide</td>
<td>Divides one series by another.</td>
</tr>
<tr>
<td>Multiply</td>
<td>Multiplies two series together.</td>
</tr>
<tr>
<td>Average</td>
<td>Calculates the average of a single series.</td>
</tr>
</tbody>
</table>
Creating Custom Calculations

You can create functions in JavaScript that perform custom calculations on trend series. To call the JavaScript function, you need to add a custom function to the trend, and configure it to use the JavaScript function name using the `setJSFunction` method.

You can give the JavaScript function any name. The function receives the function object as a parameter when it is called. Using the function object’s methods, the JavaScript function can perform the calculation and return the result.

### Function Name | Description
--- | ---
**Min** | Calculates the minimum value of a series.
**Max** | Calculates the maximum value of a series.
**Moving Average** | Calculates the moving average of a series.
**Std Deviation** | Calculates the standard deviation of a series.
**X-Y Plot** | Plots the Y values of one series against the Y values of another.
**Custom** | Runs a JavaScript function to perform the calculation.
The function object exposes the methods shown in the following table.

**Methods Available Using setJSFunction**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getIsLastValue</td>
<td>Returns true if the current value is the last value is the series.</td>
</tr>
<tr>
<td>getIsFirstValue</td>
<td>Returns true if the current value is the first value in the series.</td>
</tr>
<tr>
<td>getVal1</td>
<td>Returns the current value for the first series.</td>
</tr>
<tr>
<td>getVal2</td>
<td>Returns the current value for the second series, if defined.</td>
</tr>
<tr>
<td>getOwnerSeries</td>
<td>Returns the function series object for this calculation.</td>
</tr>
<tr>
<td>getMinX</td>
<td>Returns the smallest X value for the source series.</td>
</tr>
<tr>
<td>getMaxX</td>
<td>Returns the largest X value for the source series.</td>
</tr>
</tbody>
</table>

**Example: Adding a custom function (CustomCalc1.html).**

```html
<HTML>
  <HEAD>
    <SCRIPT defer>
    function btnSeries.onclick()
    {
      var ctChart = PNJTrend.getChart();
      var csSeries = ctChart.addSeries("My Series", true);
      csSeries.addXYStatus(1, 1, 192);
      csSeries.addXYStatus(2, 2, 192, "Some Text");
      csSeries.addXYStatus(3, 3, 192);
      var csFunc = ctChart.addFunctionSeries("Func", "Custom");
      csFunc.setSourceSeries("My Series");
      csFunc.getFunction().setUnary(true);
      csFunc.getFunction().setJSFunction("timesTen");
      PNJTrend.repaint();
    } //btnSeries.onclick

    function timesTen(oFunc)
    {
      return oFunc.getVal1() * 10;
    } //timesTen
    </SCRIPT>
  </HEAD>
  <BODY>
    <APPLET id=PNJTrend
      code=PNJTrend.BasicTrend
      archive="/ProcessNet/HTML/PNJTrend/PNJTrend.jar"
      width=100%
      height=80%
      MAYSSCRIPT>
    </APPLET>
    <BUTTON id=btnSeries>Add Series</BUTTON>
  </BODY>
</HTML>
```
Chapter 9: NetTrend Programming Interface

Example: Adding a custom function that calculates over the full range of data (CustomCalc2.html).

```
<SCRIPT defer>
function btnSeries.onclick()
{
    var ctChart = PNJTrend.getChart();
    var csSeries = ctChart.addSeries("My Series", true);
    csSeries.addXYStatus(1, 1, 192);
    csSeries.addXYStatus(2, 2, 192, "Some Text");
    csSeries.addXYStatus(3, 3, 192);
    var csFunc = ctChart.addFunctionSeries("Func", "Custom");
    csFunc.setSourceSeries("My Series");
    csFunc.setDescriptor("Average Minus 10");
    csFunc.getFunction().setUnary(true);
    csFunc.getFunction().setJSFunction("averageMinusTen");
    PNJTrend.repaint();
}
</btnSeries.onclick

var fTotal = 0;
var nCount = 0;

function averageMinusTen(oFunc)
{
    if( oFunc.getIsFirstValue() )
    {
        fTotal = 0;
        nCount = 0;
    }
    fTotal += oFunc.getVal1();
    nCount ++;
    var fAvg = fTotal/nCount - 10;
    // if last value, calculate the average and plot
    if( oFunc.getIsLastValue() )
    {
        var csSeries = oFunc.getOwnerSeries();
        csSeries.clear();
        csSeries.AddXYStatus(oFunc.getMinX(), fAvg, 192);
    }
    return fAvg;
}
</SCRIPT>
</HEAD>

Adding Constants

You can add a constant line to the trend that obtains its value from the Operational Insight server, or has a value specified for it when it is added. To add a constant to the trend, you need to use the `addSeriesFromClass` method in the chart. The
**addSeriesFromClass** method takes the Java class name for the series and the series title as parameters. The Java class name for a constant series is PNJTrend.SeriesBand.

**Example: Adding a constant series with a fixed value** ([FixedConstant.html](#)).

```html
<HTML>
  <HEAD>
    <SCRIPT defer>
      function btnSeries.onclick()
      {
        var ctChart = PNJTrend.getChart();
        var csSeries =
          ctChart.addSeriesFromClass("PNJTrend.SeriesBand", "MySeries");
        csSeries.setConstValue(100);
        csSeries.setHeight(0);
        PNJTrend.repaint();
      }//btnSeries.onclick
    </SCRIPT>
  </HEAD>
  <BODY>
    <APPLET id=PNJTrend
      code=PNJTrend.BasicTrend
      archive = "/ProcessNet/HTML/PNJTrend/PNJTrend.jar"
      width = 100%
      height = 80%
     MAYSRIPT>
    </APPLET>
    <BUTTON id=btnSeries>Add Series</BUTTON>
  </BODY>
</HTML>
```

**Example: Adding a constant series with a value from a relational database** ([QueryConstant.html](#)).

```html
<HTML>
  <HEAD>
    <SCRIPT defer>
      function btnSeries.onclick()
      {
        var sPath = "/ProcessNet/isapi/netportal/netportal.dll/data/";
        var ctChart = PNJTrend.getChart();
        var csSeries =
          ctChart.addSeriesFromClass("PNJTrend.SeriesBand", "MySeries");
        csSeries.setHeight(0);
        csSeries.setHistorySource(sPath+"SQL_DS");
        csSeries.setHistTag("select saledate, itemstotal, 192 from orders");
        ctChart.getDSManager().registerSeries(csSeries, true);
      }//btnSeries.onclick
    </SCRIPT>
  </HEAD>
  <BODY>
    <APPLET id=PNJTrend
      code=PNJTrend.BasicTrend
      archive = "/ProcessNet/HTML/PNJTrend/PNJTrend.jar"
      width = 100%
      height = 80%
     MAYSRIPT>
    </APPLET>
    <BUTTON id=btnSeries>Add Series</BUTTON>
  </BODY>
</HTML>
```
Adding Fill Plots

The Operational Insight Trend supports Fill Plots, which can map to any type of series. You can add a Fill Plot to the trend using the `addSeriesFromClass` methods, as described in “Adding Constants” on page 114. The Java class name for the Fill Plot is `PNJTrend.FillPlotSeries`. The Fill Plot inherits all methods of a basic series. You can map a color and status between a low and upper bound range corresponding to data plotted by the source series.

The `addRange` method is used to define a color and status for the specified range.

**Parameters for Adding a Fill Plot to a Series**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sColor</td>
<td>string</td>
<td>The hexadecimal string corresponding to the RGB color (0xRRGGBB).</td>
</tr>
<tr>
<td>sStatus</td>
<td>string</td>
<td>The status label for this range.</td>
</tr>
<tr>
<td>dLow</td>
<td>double</td>
<td>The lower bound of the range.</td>
</tr>
<tr>
<td>dHigh</td>
<td>double</td>
<td>The upper bound of the range.</td>
</tr>
</tbody>
</table>

**Example: Adding a Fill Plot to a series (FillPlot.html)**

```html
<HTML>
<HEAD>
<SCRIPT defer>
function btnFillPlot.onclick()
{
 var sPath = "/ProcessNet/isapi/netportal/netportal.dll/data/";
 var ctChart = PNJTrend.getChart();

 // Create a Series the Fill Plot will be based off //
 // Create a Fill Plot based of the series Saw-toothed Waves.Real8 //
 var csFillPlot =
 ctChart.addSeriesFromClass("PNJTrend.FillPlotSeries", "Fill - " +
 csSeries.getTitle());
 csFillPlot.setDSFunction( csSeries.getDSFunction() );
 csFillPlot.setStep( csSeries.getStep() );
 csFillPlot.setHistorySource( csSeries.getHistorySource() );
 csFillPlot.setHistTag( csSeries.getHistTag() );
 csFillPlot.setRealTimeSource( csSeries.getRealTimeSource() );
 sFillPlot.setRTTag( csSeries.getRTTag() );

 // Set the Timing for the Fill Plot relating to Saw-toothed Waves.Real8
 csFillPlot.setStartTime( csSeries.getStartTime() );
 csFillPlot.setEndTime( csSeries.getEndTime() );
 csFillPlot.setRTRefresh( csSeries.getRTRefresh() );
```

*Operational Insight R362.1 Programmer's Guide*
// Define Various Color / Status Ranges mapping to Saw-toothed Waves.Real8
ctFillPlot.addRange( "0xFF0000", "OFF", 0, 50);
ctFillPlot.addRange( "0x770000", "ON", 50, 100);
ctFillPlot.addRange( "0x007700", "RUNNING", 100, 150);
ctFillPlot.addRange( "0x00FF00", "OPTIMAL", 150, 200);
ctFillPlot.addRange( "0x999999", "BLOWN", 200, 250);
ctFillPlot.setActive(true);

// Register the Fill Plot
ctChart.getDSManager().registerSeries(ctFillPlot, true);
ctChart.getDSManager().getData();
}

// btnFillPlot.onclick
</SCRIPT>
</HEAD>
<BODY>
<APPLET id=PNJTrend
width=100%
height=80%
MAYSCRIPT>
</APPLET>
</BODY>
</HTML>

Adding Bars

In addition to line charts, the Operational Insight trend also supports bar charts. You can add a bar series to the trend using the addSeriesFromClass method, as described in “Adding Constants” on page 114. The Java class name for a bar series is PNJChart.SeriesBar.

Example: Adding a bar series with data from a relational database (BarSeries.html).

<html>
<head>
<script defer>
function btnSeries.onclick()
{
 var sPath = 
 "/ProcessNet/isapi/netportal/netportal.dll/data/";
 var ctChart = PNJTrend.getChart();
 var csSeries =
 ctChart.addSeriesFromClass("PNJTrend.SeriesBar", 
 "Bar Series");
 csSeries.setHistorySource(sPath+"SQL_DS");
 csSeries.setHistTag("select VenueNo, Capacity, 192 from venues");
 csSeries.XAxis.setDisplayFormat("0");
 csSeries.XAxis.setCursorDisplayFormat("0.0");
 ctChart.getDSManager().registerSeries(csSeries,true);
}
</script>
</head>
<body>
</body>
</html>
Adding Axes

The Operational Insight trend supports multiple moveable axes. You can add and position these axes programmatically. The trend provides two axis types by default: horizontal and vertical. You can access the axes through the chart’s axis list object, which also provides methods for adding and removing axes.

When setting the position of an axis on the trend, you must modify one or more of the following parameters shown in the following table.

Parameters for Positioning Axes on Trends

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Axis Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td>The starting position of the axis in percent. For Y axes, a start of zero starts drawing the axis at the top of the trend. A start of 100 draws the top of the axis at the bottom of the trend. For X axes, a start of zero starts drawing the left of the axis at the very left of the trend. A start of 100 draws the left of the axis at the right of the trend.</td>
<td>setStart()</td>
</tr>
<tr>
<td>End</td>
<td>The ending position of the axis in percent. For Y axes, an end of 100 draws the bottom of the axis at the bottom of the trend. For X axes, an end of 100 draws the right of the axis at the right of the trend.</td>
<td>setEnd()</td>
</tr>
</tbody>
</table>
### Example: Adding a vertical axis (VertAxis.html).

```html
<HTML>
<HEAD>
  <SCRIPT defer>
  function btnSeries.onclick()
  {
    var sPath = 
      "/ProcessNet/isapi/netportal/netportal.dll/data/";
    var ctChart = PNJTrend.getChart();
    ctChart.addSeries("Random 1", sPath+"OPC_SIM", 
      "Random.Int1", "*-10M", 
      "Random.Int1", 1);
    var csSeries = ctChart.addSeries("Random 2", 
      sPath+"OPC_SIM", "Random.Int2", "*-10M", 
      "Random.Int2", 1);
    var axAxis = ctChart.getAxes().addVertAxis("Right");
    axAxis.setPositionPct(100); 
    csSeries.YAxis = axAxis;
    ctChart.getDSManager().getData();
  }
  </SCRIPT>
</HEAD>

<BODY>
  <APPLET id=PNJTrend
    code=PNJTrend.BasicTrend
    archive = "/ProcessNet/HTML/PNJTrend/PNJTrend.jar"
    width = 100%
    height = 80%
    MARGINS>
  </APPLET>
  <BUTTON id=btnSeries>Add Series</BUTTON>
</BODY>
</HTML>
```

### Example: Adding a horizontal axis (HorizAxis.html).

```html
<HTML>
<HEAD>
  <SCRIPT defer>
  function btnSeries.onclick()
  {
    var ctChart = PNJTrend.getChart();
    var csSeries1 = ctChart.addSeries("My Series", true);
    csSeries1.addXYStatus(1, 1, 192); 
    csSeries1.addXYStatus(2, 2, 192); 
    csSeries1.addXYStatus(3, 3, 192); 
    var csSeries2 = ctChart.addSeries("My Series2", true);
    csSeries2.addXYStatus(101, 3, 192);
    csSeries2.addXYStatus(102, 2, 192);
  }
  </SCRIPT>
</HEAD>

```
Trend Events

The trend can notify the underlying HTML page when specific events occur, such as the user clicking the trend. The trend notifies the underlying page by calling a JavaScript function with a specific name. The following sections describe the events that are supported by the trend.

OnClick

The **OnClick** event is generated when the user clicks an area of the trend. This event is not generated when the user clicks on a series, axis, or legend. The JavaScript function that the trend calls is defined by the `OnClick` parameter of the trend. The function receives the parameters shown in the following table.

**Parameters for OnClick Function**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>trTrend</td>
<td>The trend object generating the event.</td>
</tr>
<tr>
<td>nXPos</td>
<td>The X position of mouse when clicked.</td>
</tr>
<tr>
<td>nYPos</td>
<td>The Y position of mouse when clicked.</td>
</tr>
<tr>
<td>bWasRight</td>
<td>True if the right button was clicked.</td>
</tr>
</tbody>
</table>

Example: Handling the OnClick event (**OnClick.html**).

```html
<HTML>
  <HEAD>
    <SCRIPT>
    function TrendOnClick(trTrend, nXPos, nYPos, bWasRight) {
```

```javascript
    var axAxis = ctChart.getAxes().addHorzAxis("Right");
    axAxis.setStart(51);
    axAxis.setDisplayFormat("0.0");
    axAxis.setCursorDisplayFormat("0.0");
    csSeries1.XAxis.setEnd(49);
    csSeries1.XAxis.setDisplayFormat("0.0");
    csSeries1.XAxis.setCursorDisplayFormat("0.0");
    csSeries2.XAxis = axAxis;
    PNJTrend.repaint();
  }
```

```html
  </SCRIPT>
  </HEAD>
</HTML>

```html
<APPLET id=PNJTrend
  code=PNJTrend.BasicTrend
  archive = "/ProcessNet/HTML/PNJTrend/PNJTrend.jar"
  width = 100%
  height = 80%
  MAYSRIPT>
</APPLET>
```
OnSeriesClick

The OnSeriesClick event is generated when the user clicks a trend marker. The trend calls a JavaScript function defined by the OnSeriesClick parameter of the trend. This function receives the parameters shown in the following table.

### Parameters for OnSeriesClick Function

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>trTrend</td>
<td>The trend object generating the event.</td>
</tr>
<tr>
<td>csSeries</td>
<td>The series that was clicked.</td>
</tr>
<tr>
<td>nXPos</td>
<td>The X position of the mouse when clicked.</td>
</tr>
<tr>
<td>nYPos</td>
<td>The Y position of the mouse when clicked.</td>
</tr>
<tr>
<td>bWasRight</td>
<td>True if the right button was clicked.</td>
</tr>
<tr>
<td>nClickCount</td>
<td>The number of times the mouse was clicked (1 or 2). This is useful for determining if the point was double-clicked or not.</td>
</tr>
<tr>
<td>nIndex</td>
<td>The index for the marker that was clicked.</td>
</tr>
<tr>
<td>nXPosTrend</td>
<td>The trend X position of the mouse when clicked.</td>
</tr>
<tr>
<td>nYPosTrend</td>
<td>The trend Y position of the mouse when clicked.</td>
</tr>
</tbody>
</table>

Example: Handling the OnSeriesClick event (OnSeriesClick.html).

```html
<HTML>
<!-- HEAD -->

<SCRIPT>
function onSeriesClick(trTrend,csSeries, nXPos, nYPos,
                        bWasRight, nClickCount, nIndex, nXPosTrend, nYPosTrend)
{
    alert("Series was clicked at "+nXPos+","+nYPos);
}
</SCRIPT>
</HEAD>

<APPLET id=PNJTrend
        code=PNJTrend.BasicTrend
        archive = "/ProcessNet/HTML/PNJTrend/PNJTrend.jar"
        width = 100%
        height = 100%
        MAYSCRIPT>
        <PARAM name=OnClick value=TrendOnClick>
    </APPLET>
</BODY>
</HTML>
```
function buildseries()
{
  var csSeries = PNJTrend.getChart().addSeries("MySeries",
      true);
  csSeries.AddXYStatus(1,1,192);
  csSeries.AddXYStatus(2,2,192);
  csSeries.AddXYStatus(3,3,192);
  PNJTrend.repaint();
}

</SCRIPT>
</HEAD>
<BODY>
<APPLET id=PNJTrend
  code=PNJTrend.BasicTrend
  archive = "/ProcessNet/HTML/PNJTrend/PNJTrend.jar"
  width = 100%
  height = 90%
  MAYSRIPT>
</APPLET>
<BUTTON onclick="buildseries();">Build Series</BUTTON>
</BODY>
</HTML>

OnDoubleClick

The OnDoubleClick event is generated when the user double-clicks an area of
the trend. This event is not generated when the user clicks on a series, axis, or
legend. The JavaScript function that the trend calls is defined by the OnDblClick
parameter of the trend. If the name of the function is not specified (shown in the
following example specifying TrendOnDblClick), it defaults to
TrendDblClick. This function receives a single parameter: a string representing
the trend's ID. This ID is set with the applet's ID parameter.
Example: Handling the OnDoubleClick event (OnDoubleClick.html).

<HTML>
<HEAD>
<SCRIPT>
  function TrendOnDblClick(sID)
  {
    alert(sID+" was clicked");
  }
</SCRIPT>
</HEAD>
<BODY>
<APPLET id=PNJTrend
  code=PNJTrend.BasicTrend
  archive = "/ProcessNet/HTML/PNJTrend/PNJTrend.jar"
  width = 100%
  height = 100%
  MAYSRIPT>
<PARAM name=OnDblClick value=TrendOnDblClick >
<PARAM name=ID value=Trend1>
</APPLET>
</BODY>
</HTML>

OnError

The trend generates the OnError event whenever an unexpected exception
occurs within the trend. This includes failure to retrieve data and invalid
configuration settings.
The trend calls a JavaScript function defined by the OnError parameter of the trend. This function receives the parameters shown in the following table.

**Parameters for OnError Function**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sDesc</td>
<td>A brief description of the error.</td>
</tr>
<tr>
<td>sMsg</td>
<td>A detailed error message.</td>
</tr>
<tr>
<td>trTrend</td>
<td>The trend object generating the event.</td>
</tr>
</tbody>
</table>

Example: Handling the OnError event (OnError.html).

```html
<HTML>
  <HEAD>
    <SCRIPT>
      function onError(sDesc, sMsg)
      {
        alert(sDesc+"":"+sMsg);
      }
    </SCRIPT>
  </HEAD>
  <BODY>
    <APPLET id=PNJTrend
      code=PNJTrend.BasicTrend
      archive = "/ProcessNet/HTML/PNJTrend/PNJTrend.jar"
      width = 100%
      height = 100%
      MAYSRCPT>
      <PARAM name=OnError value=onError >
    </APPLET>
  </BODY>
</HTML>
```

This example shows how to handle the OnError event, but does not include an error.

**OnZoom**

The OnZoom event is generated immediately after the user finishes zooming into a region of the trend, but before the trend is repainted. In order to handle this event, a JavaScript function named onZoom (note that the “o” in “on” is lowercase) must be present in the underlying HTML document. The onZoom function is passed to the trend object as a parameter.

Example: Handling the onZoom event (OnZoom.html).

```html
<HTML>
  <HEAD>
    <SCRIPT>
      function onZoom(oTrend)
      {
        alert("Trend zoomed");
      }
    </SCRIPT>
  </HEAD>
  <BODY>
    <APPLET id=PNJTrend
      code=PNJTrend.BasicTrend
      archive = "/ProcessNet/HTML/PNJTrend/PNJTrend.jar"
      width = 100%
      height = 100%
      MAYSRCPT>
      <PARAM name=OnZoom value=onZoom >
    </APPLET>
  </BODY>
</HTML>
```
OnUnZoom

The trend generates the OnUnZoom event when the user un-zooms the trend, before the trend is repainted. The OnUnZoom event is handled in the same way as the OnZoom event: you must create a function called onUnZoom which is called by the trend, and passed a reference to the trend object.

OnTrendLoaded

The trend generates the OnTrendLoaded event during the init() method of the applet. Since the Microsoft Virtual Machine (VM) calls init(), start(), and then init() again, this event is generated twice under the Microsoft VM, and once under the Sun VM. You must provide a function called OnTrendLoaded that is called by the trend. This function receives the parameters shown in the following table.

Parameters for onLegendClick Function

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>trTrend</td>
<td>The trend object generating the event.</td>
</tr>
<tr>
<td>sID</td>
<td>The string representing the trend’s ID.</td>
</tr>
</tbody>
</table>

OnDataLoaded

The trend generates the OnDataLoaded event when:
- All data for all historical series have been loaded.
- All data for any “real-time only” series is loaded, regardless of how many “real-time only” series there are.
- A new series is added and the data retrieved.

The purpose of the OnDataLoaded event is to provide notification to objects that the trend has all the required data on it, and is ready to be manipulated. The Trend Control Bar uses this event to initialize itself.

You must provide a function called OnDataLoaded that is called by the trend. As with the OnZoom event, the trend passes a reference to itself to the function.

OnLegendClick

The trend generates the OnLegendClick event whenever the user clicks the legend. You must provide a function called onLegendClick that is called when the user clicks the legend. This function receives the parameters in the following table.
Parameters for `onLegendClick` Function

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>trTrend</td>
<td>The trend object generating the event.</td>
</tr>
<tr>
<td>csSeries</td>
<td>The series that was clicked in the legend.</td>
</tr>
<tr>
<td>nXPos</td>
<td>The X position of the mouse when clicked.</td>
</tr>
<tr>
<td>nYPos</td>
<td>The Y position of the mouse when clicked.</td>
</tr>
<tr>
<td>bWasRight</td>
<td>True if the right button was clicked.</td>
</tr>
</tbody>
</table>

**OnLegendMouseMove**

The trend generates the `OnLegendMouseMove` event whenever the user moves the mouse on the legend. You must provide a function called `onLegendMouseMove` that is called when the user moves the mouse on the legend. This function receives the parameters shown in the following table.

Parameters for the `onLegendClick` Function

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>trTrend</td>
<td>The trend object generating the event.</td>
</tr>
<tr>
<td>csSeries</td>
<td>The series over which the mouse was moved in the legend.</td>
</tr>
<tr>
<td>nXPos</td>
<td>The X position of the mouse when moved.</td>
</tr>
<tr>
<td>nYPos</td>
<td>The Y position of the mouse when moved.</td>
</tr>
</tbody>
</table>

**OnAxisClick**

The trend generates the `OnAxisClick` event when the user clicks an axis. You must provide a function called `onAxisClick` to handle this event. The `onAxisClick` function receives the parameters shown in the following table.

Parameters for the `onAxisClick` Function

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>trTrend</td>
<td>The trend object generating the event.</td>
</tr>
<tr>
<td>axAxis</td>
<td>The axis object that was clicked.</td>
</tr>
<tr>
<td>nXPos</td>
<td>The X position of the mouse when clicked.</td>
</tr>
<tr>
<td>nYPos</td>
<td>The Y position of the mouse when clicked.</td>
</tr>
<tr>
<td>bWasRight</td>
<td>True if the right mouse button was clicked.</td>
</tr>
</tbody>
</table>
OnCursor

The trend generates the OnCursor event when a cursor is added or dropped. A cursor is dropped whenever the user drags it somewhere and releases the mouse. Using this event and several methods on the axis and cursor objects, you can determine the position of the cursor.

The onCursor function receives the parameters shown in the following table.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>trTrend</td>
<td>The trend object generating the event.</td>
</tr>
<tr>
<td>crCursor</td>
<td>The cursor object that was dropped or added.</td>
</tr>
</tbody>
</table>

Example: Handling the OnCursor event (OnCursor.html).

```html
<HTML>
<HEAD>
<SCRIPT>
<!-- Code to add data to trend required -->
function onCursor(oTrend, crCursor)
{
    var axAxis = PNJTrend.getChart().getDfltXAxis();
    var nPixelPos = crCursor.getPosition();
    alert( axAxis.GetValueAtPos(nPixelPos) );
}
</SCRIPT>
</HEAD>
<BODY>
<APPLET id=PNJTrend
code=PNJTrend.BasicTrend
archive="/ProcessNet/HTML/PNJTrend/PNJTrend.jar"
width=100%
height=100%
MAYSCRIPT>
</APPLET>
</BODY>
</HTML>
```

OnDeleteCursor

The trend generates the onDeleteCursor event when the user removes a cursor by dragging it off the side of the trend.

The onDeleteCursor function receives the parameters shown in the following table.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>trTrend</td>
<td>The trend object generating the event.</td>
</tr>
<tr>
<td>crCursor</td>
<td>The cursor object that was deleted.</td>
</tr>
</tbody>
</table>
Trend Events Summary

The following table summarizes the parameters and functions that are used with trend events.

Trend Events Parameters and Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Applet Parameter</th>
<th>JavaScript Function</th>
<th>Function Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>OnClick</td>
<td>OnClick</td>
<td>trTrend, nXPos, nYPos, bWasRight</td>
<td></td>
</tr>
<tr>
<td>OnSeriesClick</td>
<td></td>
<td>OnSeriesClick</td>
<td>trTrend, csSeries, nXPos, nYPos, bWasRight, nClickCount, nIndex, nXPosTrend, nYPosTrend</td>
</tr>
<tr>
<td>OnDoubleClick</td>
<td>OnDBlClick</td>
<td>TrendDblClick</td>
<td>sID</td>
</tr>
<tr>
<td>onError</td>
<td>OnError</td>
<td></td>
<td>sdDesk, sMsg, trTrend</td>
</tr>
<tr>
<td>OnZoom</td>
<td></td>
<td>OnZoom</td>
<td>trTrend</td>
</tr>
<tr>
<td>OnUnZoom</td>
<td></td>
<td>OnUnZoom</td>
<td>trTrend</td>
</tr>
<tr>
<td>OnDataLoaded</td>
<td></td>
<td>OnDataLoaded</td>
<td>trTrend</td>
</tr>
<tr>
<td>OnTrendLoaded</td>
<td></td>
<td>OnTrendLoaded</td>
<td>trTrend, sID</td>
</tr>
<tr>
<td>OnLegendClick</td>
<td></td>
<td>OnLegendClick</td>
<td>trTrend, csSeries, nXPos, nYPos, bWasRight</td>
</tr>
<tr>
<td>OnLegendMouseMove</td>
<td></td>
<td>OnLegendMouseMove</td>
<td>trTrend, csSeries, nXPos, nYPos</td>
</tr>
</tbody>
</table>
The preceding sections describe only portions of the programming interface. For more information, look at the trend examples installed in the Operational Insight tree in the Public folder, and refer to the JavaDocs, located in the javadocs folder on the installation CD.
Chapter 10: Other Operational Insight Objects

Operational Insight uses several objects to complete the default user interface. You can use these objects in your own pages and applications. The following sections describe these objects and discuss how they can be used.

DataHub Object

The DataHub is a JavaScript object that provides an easy way to perform data requests at a defined interval. You can use the DataHub to update custom displays and trends. To use the DataHub, you must include the proper script file:

```html
<script
    language=JScript.Encode
    src=/ProcessNet/netdraw/script/datahub.js>
</script>
```

The DataHub is a “singleton” object, which means that only one instance of it will ever exist. To ensure that multiple DataHub objects are not created, the DataHub.js script file provides a function to retrieve the DataHub object. The getHubObject method takes a WebDataSet object as a parameter, and returns the DataHub object.

The DataHub provides the methods shown in the following table.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>addQuery</td>
<td>Defines a data request and an event function that is to be called periodically.</td>
</tr>
<tr>
<td>bNoQueryOnBegin</td>
<td>A Boolean method. If true, the first data query will not occur immediately when start is called, but rather one interval after.</td>
</tr>
<tr>
<td>start</td>
<td>Starts data access.</td>
</tr>
<tr>
<td>stop</td>
<td>Stops data access.</td>
</tr>
<tr>
<td>bPause</td>
<td>A Boolean method. If true, the queries are not deleted from the DataHub, allowing them to be resumed by a subsequent call to start.</td>
</tr>
</tbody>
</table>
The `addQuery` method takes the parameters shown in the following table.

### Parameters for `addQuery` Function

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>fnFunc</code></td>
<td>Function</td>
<td>The JavaScript function to call when the data for the query is returned.</td>
</tr>
<tr>
<td><code>oParams</code></td>
<td>Object</td>
<td>A JavaScript object that is passed to the event function defined by <code>fnFunc</code>.</td>
</tr>
<tr>
<td><code>sDS</code></td>
<td>String</td>
<td>The Operational Insight data source name, without the NetPortal path.</td>
</tr>
<tr>
<td><code>sQuery</code></td>
<td>String</td>
<td>The data access command, as defined in the URL query section.</td>
</tr>
<tr>
<td><code>nInterval</code></td>
<td>Integer</td>
<td>The delay, in milliseconds, between data requests.</td>
</tr>
<tr>
<td><code>bAsynchronous</code></td>
<td>Boolean</td>
<td>If true, the data request is non-blocking.</td>
</tr>
</tbody>
</table>

The function passed to the `addQuery` method is called when the data request is made. The DataHub passes a Recordset object and the parameter defined by `oParams` in the call to `addQuery`.

The DataHub will call a function named `doAlert` if an error occurs, and will pass it one parameter: the string representing the error message.

**Note**

As of Operational Insight 3.2, if a function named `doAlert` does not exist, then the error message string will be passed to the JavaScript `alert` method, and the error string will be displayed in an alert message box.

**Example: Using the DataHub (`DataHub.html`).**

```html
<html>
<head>
  <script language=JScript.Encode src=/ProcessNet/netdraw/script/datahub.js>
  </script>
  <script defer=
    function btnStart.onclick() {
      var rtHub = getHubObject(PNWebDS);
      rtHub.addQuery(gotTagData, divRandom, "OPC_SIM", "Tag0=Random.Int2", 5000);
      rtHub.addQuery(gotTagData, divSawtooth, "OPC_SIM", "Tag0=Saw-toothed waves.Int2", 1000);
      rtHub.addQuery(gotSQLData, "", "SQL_DS", "SQL=select Custno from orders", 30000);
      rtHub.start();
    }
  </script>
</head>
```

---

Chapter 10: Other Operational Insight Objects
Menu Applet

Operational Insight provides a simple menu that can be used either for context menus or as a main menu, as in NetTrend. The menu object is a Java applet, enabling it to be shown above other Java applets, such as the trend. The menu applet also supports pop-up menus.

The menu is included in the HTML page with an `<APPLET>` element:

```html
<APPLET id='MenuApp' 
    code='MenuApp.MenuApplet.class' 
    archive='ProcessNet/html/pnjTrend/pnjTrend.jar' 
    style='position:absolute;top:0;left:0;height:1;width:1'><MAYSCRIPT>
</APPLET>
```

The menu applet takes the parameters shown in the following table.
Menu Applet Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delimiter</td>
<td>String</td>
<td>Defaults to a comma. This is the character used to separate menu items in function parameters, such as sItems.</td>
</tr>
<tr>
<td>bgColor</td>
<td>String</td>
<td>Defaults to the background color of the window. If specified, the applet will be this color. This is typically not used, because the applet is only one pixel in size.</td>
</tr>
</tbody>
</table>

The menu object has a method called `ShowMenu`. This method takes the parameters shown in the following table.

Parameters for ShowMenu Method

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nX</td>
<td>Integer</td>
<td>The X position to show the menu, relative to the position of the applet. If the applet is positioned at 0,0, then the X position can be treated like an absolute position.</td>
</tr>
<tr>
<td>nY</td>
<td>Integer</td>
<td>The Y position to show the menu, relative to the position of the applet. If the applet is positioned at 0,0, then the Y position can be treated like an absolute position.</td>
</tr>
<tr>
<td>sItems</td>
<td>String</td>
<td>A comma-separated list of the menu items you want to display on the menu.</td>
</tr>
<tr>
<td>sEvent</td>
<td>String</td>
<td>The name of the JavaScript function you want to call when the menu is clicked. The text of the item that was clicked is passed to this function.</td>
</tr>
</tbody>
</table>

Example: Using the menu object to create a right-click (context) menu (MenuApp.html).

```html
<HTML>
<HEAD>
<SCRIPT>
function doShowMenu() {
    MenuApp.ShowMenu(event.x, event.y, "Option 1,Option 2,Option 3", "onMenuClick");
    return false;
} //doShowMenu()

function onMenuClick(sItem) {
    switch(sItem) {
```
Calendar Control

Calendar Control is replaced with jQueryUI datepicker control. To use datepicker control and its API check the following URL for the demo code and API documentation.
http://jqueryui.com/datepicker/

Example for using Calendar control (Calendar.html)

```html
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>jQuery UI Datepicker - Default functionality</title>
  <link href="/processnet/html/jquery-ui/themes/base/minified/jquery-ui.css" rel="stylesheet" type="text/css" />
  <script src="/ProcessNet/html/jquery-ui/jquery.js" type="text/javascript"></script>
  <script>
    $(function() {
      $('#datepicker').datepicker();
    });
  </script>
</head>
<body>

<p>Date: <input type="text" id="datepicker"></p>

</body>
</html>```
Chapter 10: Other Operational Insight Objects

Trend Control Bar

The Trend control bar can be used to:
- Control the time ranges of trend axes.
- Refetch data on an axis or axes.
- Auto-arrange or create axes based on the trend data.

This control can be applied to virtually any standard HTML element in the form of an HTC (HTML Component) behavior (`PNTrendCtrlBar.htc`).

The following properties can only be defined inline with the control bar’s HTML definition. If they are changed during run time using script, the changes are not applied.

**Inline Parameters for Trend Control Bar**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>align</code></td>
<td>The alignment of the control bar.</td>
<td><code>left</code></td>
</tr>
<tr>
<td><code>contentalign</code></td>
<td>The alignment of the control bar's controls within its parent container.</td>
<td><code>center</code></td>
</tr>
<tr>
<td><code>width</code></td>
<td>The width of the control bar.</td>
<td><code>0%</code></td>
</tr>
<tr>
<td><code>border</code></td>
<td>The border style.</td>
<td><code>1px solid outset</code></td>
</tr>
<tr>
<td><code>bgcolor</code></td>
<td>The background color.</td>
<td><code>buttonface</code></td>
</tr>
<tr>
<td><code>buttoncolor</code></td>
<td>The color you want to use for the buttons.</td>
<td><code>buttonface</code></td>
</tr>
<tr>
<td><code>color</code></td>
<td>The color you want to use for the text.</td>
<td><code>windowtext</code></td>
</tr>
<tr>
<td><code>inputbgcolor</code></td>
<td>The background color of the inputs.</td>
<td><code>window</code></td>
</tr>
<tr>
<td><code>inputwidth</code></td>
<td>The width of the input controls.</td>
<td><code>125</code></td>
</tr>
<tr>
<td><code>font</code></td>
<td>The font you want to use on the control bar.</td>
<td><code>Arial</code></td>
</tr>
<tr>
<td><code>imgsrcpath</code></td>
<td>The path from which to load the images.</td>
<td><code>/ProcessNet/HTML/Img/</code></td>
</tr>
<tr>
<td><code>id</code></td>
<td>The ID of the control bar (must be unique).</td>
<td><code>PNTrendCtrlBar</code></td>
</tr>
</tbody>
</table>
The following properties can be modified at run time using script. They can also be specified inline with the control bar’s HTML definition. Although these values can be changed at run time, it is recommended that they only be read from and not written to.

Run-Time Parameters for Trend Control Bar

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>align</td>
<td>The alignment of the control bar.</td>
<td>left</td>
</tr>
<tr>
<td>start</td>
<td>The default start time for the axis.</td>
<td>*-1d</td>
</tr>
<tr>
<td>end</td>
<td>The default end time for the axis.</td>
<td>*</td>
</tr>
<tr>
<td>refresh</td>
<td>The default refresh interval.</td>
<td>0</td>
</tr>
<tr>
<td>rtStopped</td>
<td>Specifies if real-time data is stopped.</td>
<td>False</td>
</tr>
<tr>
<td>VariancePct</td>
<td>The percentage of variance or leeway to use when using the auto-axes function.</td>
<td>30</td>
</tr>
<tr>
<td>trendobject</td>
<td>The PNJTrend applet object to be controlled by the control bar (REQUIRED).</td>
<td>null</td>
</tr>
<tr>
<td>menuobject</td>
<td>The MenuApplet applet used by the control bar for the auto-axis menu options (REQUIRED).</td>
<td>null</td>
</tr>
</tbody>
</table>

The functions listed in the following table can be called from the control bar object using script, by appending the applet’s ID before the function call. For example:

```
PNJTrendCtrlBar.refresh();
```

Trend Control Bar Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>draw</td>
<td>Draws the control bar.</td>
</tr>
<tr>
<td>update</td>
<td>Refreshes the data displayed by the control bar.</td>
</tr>
<tr>
<td>selectAxis</td>
<td>Selects an axis when passed a zero-based axis index from the drop-down axis combo box list.</td>
</tr>
<tr>
<td>pageAxis</td>
<td>Pages the selected axis or axes forward or backward. Use true to go backward, or false to go forward.</td>
</tr>
<tr>
<td>applyTime</td>
<td>Applies the time currently entered in the control bar’s time boxes to the selected axis/axes.</td>
</tr>
</tbody>
</table>
Example: Using the control bar.

```
<DIV id = "PNTrendCtrlBar"
     menuobject = 'MenuApp' trendobject = 'PNJTrend'
     inputwidth = '100' width = "600px"
     style = 
     'behavior:url(/ProcessNet/HTML/HTC/PNTrendCtrlBar.htc);'>
</DIV>
```

**Toolbar Object**

The toolbar object is useful for providing toolbar options. The toolbar control can be defined inline with the HTML of the object, or it can be built and modified programmatically using script. This control is implemented as a behavioral HTC (HTML Component; MenuBar.htc).

The methods that apply to the toolbar object are shown in the following table.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>onMenuClick</td>
<td>Used for specifying the custom function that is called when a user clicks an item in the toolbar.</td>
<td>null</td>
</tr>
<tr>
<td>sBackgroundColor</td>
<td>Specifies the background color of the toolbar.</td>
<td>menu</td>
</tr>
<tr>
<td>sCaptionColor</td>
<td>Specifies the color of button captions.</td>
<td>MenuText</td>
</tr>
</tbody>
</table>

The methods provided by the toolbar object are shown in the following table.
Functions Provided by Toolbar Object

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>addItem</td>
<td>Adds a new menu item to the menu bar.</td>
</tr>
<tr>
<td>Enable</td>
<td>Enables or disables the specified toolbar button.</td>
</tr>
<tr>
<td>showLabels</td>
<td>Shows or hides the text captions on the toolbar.</td>
</tr>
</tbody>
</table>

Example: Using the toolbar object (`Toolbar.html`).

```html
<HTML>
<HEAD>
<TITLE>Sample Usage - Toolbar</TITLE>
<STYLE>
.qMenuItem
{
  display:none;
}
</STYLE>
<SCRIPT>
function SetupMenuButtons()
{
  var sPath = "\ProcessNet/html/img/buttons/";
  //Add a spacer
  htcMenuBar.addItem("","",false,true);
  //Add a couple of button options
  htcMenuBar.addItem(sPath+"help.gif", "About this product", "btnAbout", true, false,"About");
  htcMenuBar.addItem(sPath+"renametask.gif", "Show or hide the button labels", "btnLabels", true, false,"Labels");
  //Disable a button option
  htcMenuBar.Enable(btnOpen,false);
  htcMenuBar.Enable(btnSave,false);
}
</SCRIPT>

function MenuBar_onmenuclick(sID)
{
  switch(sID)
  {
    case "btnNew":alert("'New' was clicked'’"); break;
    case "btnOpen":alert("'Open' was clicked'’"); break;
    case "btnSave":alert("'Save' was clicked'’"); break;
    case "btnCut":alert("'Cut' was clicked'’"); break;
    case "btnCopy":alert("'Copy' was clicked'’"); break;
    case "btnPaste":alert("'Paste' was clicked'’"); break;
    case "btnAbout":alert("This is just a demo.
There is nothing else to see here!"); break;
    case "btnLabels":
      if (htcMenuBar.bShowLabels=="true")
      {
        htcMenuBar.showLabels(false);
        htcMenuBar.bShowLabels="false"
      } else
      {
        htcMenuBar.showLabels(true);
        htcMenuBar.bShowLabels="true"
      }
      break;
    default:alert("'"+sID+"' was clicked'’"); break;
  } //switch
```

Operational Insight R362.1 Programmer's Guide Page 137
Chapter 10: Other Operational Insight Objects

```html
} //MenuBar_onmenuclick
</SCRIPT>
</HEAD>
<BODY style='background-color:#C3DCF0;margin:0px;padding:0px;'
onload="SetupMenuButtons();">
<TABLE style="width:100%;height:100%">
<tr style="height:1%">
<td>
<DIV style="width:100%;height:32px;border:2 outset silver">
<DIV id="htcMenuBar"
onMenuClick="MenuBar_onmenuclick();
bShowLabels="true" sBackgroundColor="buttonface"
style="behavior:url(/ProcessNet/html/htc/MenuBar.htc)">
<DIV id="MenuItem" class="qMenuItem" isButton="true"
title="Create new document" itemID="btnNew"
sLabel="New"> /ProcessNet/html/img/buttons/new.gif
</DIV>
<DIV id="MenuItem" class="qMenuItem" isButton="true"
title="Open existing document" itemID="btnOpen"
sLabel="Open"> /ProcessNet/html/img/buttons/open.gif
</DIV>
<DIV id="MenuItem" class="qMenuItem" isButton="true"
title="Save document" itemID="btnSave" sLabel="Save">
/ProcessNet/html/img/buttons/save.gif
</DIV>
<DIV id="MenuItem" isSpacer="true">
</DIV>
<DIV id="MenuItem" class="qMenuItem" isButton="true"
title="Cut selected items" itemID="btnCut"
sLabel="Cut"> /ProcessNet/html/img/buttons/cut.gif
</DIV>
<DIV id="MenuItem" class="qMenuItem" isButton="true"
title="Copy selected items" itemID="btnCopy"
sLabel="Copy"> /ProcessNet/html/img/buttons/copy.gif
</DIV>
<DIV id="MenuItem" class="qMenuItem" isButton="true"
title="Paste items into document" itemID="btnPaste"
sLabel="Paste"> /ProcessNet/html/img/buttons/paste.gif
</DIV>
</DIV>
</DIV>
</td>
</TR>
</TABLE>
</BODY>
</HTML>
```
The ConfigFiler object lets you save and load files to and from the Operational Insight database. It also provides an interface for receiving the list of available files to which the user has access.

The ConfigFiler constructor requires two parameters, with an optional third parameter defining a custom error handling function.

The ConfigFiler constructor is:
```
ConfigFiler(sToolName, oWebDS, sErrFunc)
```

### Parameters for ConfigFiler Object

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sToolName</td>
<td>The unique tool name used for loading, saving, or getting a file list. For example, for a trend file, the tool name would be TREND.</td>
</tr>
<tr>
<td>oWebDS</td>
<td>A reference to a valid JWebDataSet applet object.</td>
</tr>
<tr>
<td>sErrFunc</td>
<td>(Optional) A string specifying a custom error function that takes only the error message as a parameter.</td>
</tr>
</tbody>
</table>

The ConfigFiler provides methods for getting access to data as shown in the following table.

### Functions Provided by ConfigFiler

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>LoadConfig</td>
<td>Loads a file from the configuration database.</td>
<td>Contents of the file.</td>
</tr>
<tr>
<td>SaveConfig</td>
<td>Saves a file to the configuration database, and associates it with the tool name specified in the ConfigFiler constructor.</td>
<td>Saved Profile ID.</td>
</tr>
<tr>
<td>ListConfig</td>
<td>Provides a listing of files associated with the specified tool name, and to which the calling user has permission. No parameters.</td>
<td>A record set with four columns: ConfigID, Title, Description, ReadOnly.</td>
</tr>
</tbody>
</table>

The LoadConfig parameters are shown in the following table.

### Parameters for LoadConfig

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sConfigID</td>
<td>The ConfigID of the file you want to load.</td>
</tr>
</tbody>
</table>
The **SaveConfig** parameters are shown in the following table.

### Parameters for SaveConfig

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sName</td>
<td>The name that will be used for the saved file.</td>
</tr>
<tr>
<td>sDescr</td>
<td>The description of the file.</td>
</tr>
<tr>
<td>sContent</td>
<td>The content to be saved.</td>
</tr>
<tr>
<td>sShortcut</td>
<td>The text for a shortcut to be placed in the tree. You can set this to null if you do not want to create a shortcut in the tree.</td>
</tr>
<tr>
<td>nFolderID</td>
<td>The ProfileID of the folder in which the tree shortcut will be placed (can be null).</td>
</tr>
<tr>
<td>nGCPID</td>
<td>The Global Config path ID to be used, if you want the file to be saved in a folder other than the user’s root folder.</td>
</tr>
<tr>
<td>sExtension</td>
<td>The default file extension that will be applied to the saved file name if required. If null, the saved file name remains unchanged.</td>
</tr>
</tbody>
</table>

**Example: Sample implementation of the ConfigFiler object.**

```javascript
<SCRIPT language = "JavaScript"
src = "/ProcessNet/HTML/SCR/ConfigFiler.js">
<SCRIPT>

/* Sample function for loading a file */
function doLoadFile(nConfigID)
{
    /* Create a configfiler object instance */
    var cfConfigFiler = new ConfigFiler("TREND", PNWebDS);
    /* Load the file */
    var sFileContents = cfConfigFiler.LoadFile(nConfigID);
    return sFileContents;
}

/* Sample function for saving a file */
function doSaveFile(sFileName, sDescription, sContent)
{
    /* Create an instance of configfiler object */
    var cfConfigFiler = new ConfigFiler("TREND", PNWebDS);
    /* Save the file. */
    var nNewProfileID = fConfigFiler.saveFile(sFileName, sDescription, sContent, null, null, null, null);
    return nNewProfileID;
}
</SCRIPT>
```

**Tabbed Dialog Box Control**

The tabbed dialog box control provides a simple framework for creating a tabbed page control. The contents and number of tabs are hard-coded, and cannot be
modified after the page has loaded. The control is implemented as a behavioral
HTC (HTML Component, TabSheet.htc).

The tabbed control dialog box has the properties shown in the following table. These are specified inline with the control’s HTML definition.

**Tabbed Control Dialog Box Properties**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>TabWidth</td>
<td>The width of the tabs on the tabbed page.</td>
<td>100px</td>
</tr>
<tr>
<td>ImgPath</td>
<td>The path from which to load the images for the tabs.</td>
<td>/ProcessNet/ html/htc/img</td>
</tr>
<tr>
<td>TabHighlightColor</td>
<td>The ‘active’ and ‘mouse-over’ color for the tab text.</td>
<td>Blue</td>
</tr>
<tr>
<td>TabFontSize</td>
<td>The font size of the tab text.</td>
<td>10pt</td>
</tr>
<tr>
<td>TabFontWeight</td>
<td>The font weight for the tab text (normal/bold).</td>
<td>Normal</td>
</tr>
<tr>
<td>TabFontColor</td>
<td>The normal/unselected font color for the tab text.</td>
<td>Black</td>
</tr>
<tr>
<td>onChange</td>
<td>Specifies a custom function to be called in the onChange event for the tab.</td>
<td>null</td>
</tr>
<tr>
<td>onBeforeChange</td>
<td>Specifies a custom function to be called before the active tab is changed.</td>
<td>null</td>
</tr>
</tbody>
</table>

The tabbed page control provides the methods shown in the following table.
Chapter 10: Other Operational Insight Objects

Tabbed Page Control Functions

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NextTab</td>
<td>Specifies that the active tab is the tab to the right of the current active tab. If the current active tab is the rightmost tab, then the leftmost tab is selected.</td>
</tr>
<tr>
<td>PreviousTab</td>
<td>Specifies that the active tab is the tab on the left of the current active tab. If the current active tab is the leftmost tab, then the rightmost tab is selected.</td>
</tr>
<tr>
<td>GotoTab</td>
<td>Specifies that the active tab is the tab indicated by a zero-based index value passed in to the function. That is, the first tab in the tab page has an index of 0.</td>
</tr>
</tbody>
</table>

Example: Using the tab page control.

```html
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
Properties for Group Box Control

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>GroupTitle</td>
<td>The text for the group box title.</td>
<td></td>
</tr>
<tr>
<td>WaitForDraw</td>
<td>If set to true, then the group box header won't draw itself until the draw function is called.</td>
<td>False</td>
</tr>
<tr>
<td>Top</td>
<td>(Optional) The absolute top position for the header.</td>
<td></td>
</tr>
<tr>
<td>Left</td>
<td>(Optional) The absolute left position for the header.</td>
<td></td>
</tr>
</tbody>
</table>

The group box control supports the method shown in the following table.

Methods for Group Box Control

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>draw</td>
<td>Draws the group box header.</td>
</tr>
</tbody>
</table>

Example: Using the group box control (GroupBox.html).

```html
<HTML>
  <HEAD>
    <LINK rel="stylesheet" type="text/css"
      href="/ProcessNet/html/dialogs/css/dialogcommon.css">
    </LINK>
  </HEAD>
  <TITLE>Group Header</TITLE>
</HEAD>
<BODY style="background-color:buttonface">
  <DIV style="border:2px groove;
    behavior:url(/ProcessNet/html/htc/groupheader.htc);"
    GroupTitle="My Group Title">
    My group content
  </DIV>
</BODY>
</HTML>
```

List Control

The list control is a behavioral HTC (HTML Component; listview.htc) that emulates some of the functionality of the Windows list view. Content for the list control is added at run-time.

This list view also requires the use of the listview.css file found in the /ProcessNet/html/htc/css path. List control properties are shown in the following table.
## List Control Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Columns</td>
<td>A comma-separated list of column headers hard-coded into the HTML. Column headers can also be set at run time using the setColumns function.</td>
<td></td>
</tr>
<tr>
<td>onChange</td>
<td>A string defining a custom function to be called when the selection in the list control changes.</td>
<td></td>
</tr>
<tr>
<td>onBeforeChange</td>
<td>A string defining a custom function to be called before the selection in the list control changes.</td>
<td></td>
</tr>
<tr>
<td>multiselect</td>
<td>If true, then multi-selection of items in the control is possible using the CTRL key.</td>
<td>true</td>
</tr>
<tr>
<td>selected</td>
<td>An array of selected list items. Each item itself is an array representing the selected item's column values. For example, to get the data in the second column of the second selected item, you would use myListview.selected[1].rowData[1];</td>
<td>null</td>
</tr>
<tr>
<td>Rows</td>
<td>A reference to the rows array used internally by the &lt;TABLE&gt; structure that makes up the list control. Because of this, Rows[0] is actually the column header, and the first element in the list is Rows[1]. For example, to get the data in the second column of the second row, you would use myListview.Rows[2].rowData[1];</td>
<td></td>
</tr>
<tr>
<td>updatetable</td>
<td>If true, then the control does not redraw itself with the addition of each item, otherwise it does. Set this to true if you are performing a large number of additions/modifications to the list control's contents. Make sure to set this back to false when you have completed your modifications, and call the refresh function.</td>
<td>false</td>
</tr>
</tbody>
</table>
List control methods are shown in the following table.

### List Control Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>setColumns</td>
<td>Adds or removes columns based on an array of column names passed into the function.</td>
</tr>
<tr>
<td>addRow</td>
<td>Adds a row to the list. Valid parameters are:</td>
</tr>
<tr>
<td></td>
<td>• aRowData: array of column information.</td>
</tr>
<tr>
<td></td>
<td>• oDataPtr: pointer to an optional object to associate with the row.</td>
</tr>
<tr>
<td></td>
<td>• sImg: optional icon to display at the beginning of the row.</td>
</tr>
<tr>
<td>refresh</td>
<td>Redraws the list control.</td>
</tr>
<tr>
<td>selectRow</td>
<td>Selects a row based on a -1 based integer value passed into the function.</td>
</tr>
<tr>
<td>unselectRow</td>
<td>Clears the row identified by the -1 based integer value passed into the function.</td>
</tr>
<tr>
<td>deleteRow</td>
<td>Deletes the row identified by the -1 based integer value passed into the function.</td>
</tr>
<tr>
<td>clear</td>
<td>Clears all items out of the list control.</td>
</tr>
<tr>
<td>selectByString</td>
<td>Selects an item from the list based on a string passed into the function.</td>
</tr>
<tr>
<td>deleteSelected</td>
<td>Removes all selected rows from the list.</td>
</tr>
</tbody>
</table>

Example: Embedding the list control in a page (List.html).

```html
<HTML>
<HEAD>
<TITLE>Untitled</title>
 LINK rel="stylesheet" type="text/css"
 href="/ProcessNet/html/htc/css/listview.css" />
</HEAD>
<SCRIPT language="JavaScript">
 function displayList()
```
Chapter 10: Other Operational Insight Objects

```javascript
{  
    var aMyList = new Array("Apples","Oranges","Bananas","Peaches","Pears");
    //add the array data to the list.
    for(var nLoop = 0; nLoop < aMyList.length;nLoop++)
    {
        MyShoppingList.addRow(new Array(nLoop, aMyList[nLoop]));
    }
}
</SCRIPT>
<BODY onLoad="displayList();">
</BODY>
</HTML>

Slider Control

The slider control is a behavioral HTC (slider.htc) that provides similar functionality as the standard windows slider control. It can be oriented either vertically or horizontally, and have any range.

Slider control properties are shown in the following table.

### Slider Control Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>The value of the control.</td>
<td>0</td>
</tr>
<tr>
<td>Vert</td>
<td>If T, then the slider control is vertical; otherwise it is horizontal (F).</td>
<td>T</td>
</tr>
<tr>
<td>Min</td>
<td>The minimum value for the control.</td>
<td>0</td>
</tr>
<tr>
<td>Max</td>
<td>The maximum value for the control.</td>
<td>0</td>
</tr>
<tr>
<td>onChange</td>
<td>The name of the function you want called when the value of the control changes.</td>
<td>null</td>
</tr>
</tbody>
</table>

The slider control has the method shown in the following table.

### Slider Control Function

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>setValue</td>
<td>Sets the value of the control to a passed integer.</td>
</tr>
</tbody>
</table>

Example: Embedding a slider control in a page.

```html
<DIV style = "behavior:url(/ProcessNet/html/htc/slider.htc);"
id = "redSlider"
Min = 0
Max = 255
Vert = "F"
```
Session Manager

The session manager is a JavaScript library (sessionmanager.js) that can be used to test the user's session, or to log on to Operational Insight without going through the logon page. This library requires that the PNDialogs.js file be included in the same page as the session manager library.

The session manager constructor requires only one variable, which is a reference to a JWebDataSet applet object. An optional second parameter is a custom error/alert function to be called if the session manager encounters an error.

The session manager methods are shown in the following table.

## Session Manager Functions

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sessionIsGood</td>
<td>Returns true if the user's session is still valid; otherwise, returns false.</td>
</tr>
<tr>
<td>doLogin</td>
<td>Performs a logon operation using the passed user name and password.</td>
</tr>
<tr>
<td>doLogout</td>
<td>Logs off any current session.</td>
</tr>
<tr>
<td>confirmSession</td>
<td>Checks the user's session, and if it is not active, displays a logon dialog box. Returns true if the session was active, or logging on was successful; otherwise, returns false.</td>
</tr>
<tr>
<td>openLoginDlg</td>
<td>Opens the logon dialog box and performs logon functions.</td>
</tr>
</tbody>
</table>

### Example: Using the session manager object (Session.html).

```html
<HTML>
  <HEAD>
    <TITLE>Session Manager</TITLE>
    <SCRIPT language="JavaScript" src="/ProcessNet/html/scr/sessionmanager.js">
    </SCRIPT>
    <SCRIPT language="JavaScript">
      function testSession() {
        var sm = new SessionManager(PNWebDS);
        //check the session and log in if neccesary
        if(!sm.confirmSession()) {
          alert("Your session is not valid, or the login process has failed.");
        } else
          alert("Your session is valid");
      }
    </SCRIPT>
</HEAD>
```
Chapter 10: Other Operational Insight Objects

Tip
Refer to these folders for other useful Javascript files:
..\ProcessNet\Web\html\scr
..\ProcessNet\Web\html\scr\lib
..\ProcessNet\Web\html\dialogs

FMI
For more information about using the Operational Insight objects described in this chapter, refer to the JavaDocs, located in the javadocs folder on the installation CD.
Chapter 11: NetDAC Server-side Data Access

Operational Insight provides programmatic access to data from server-side scripts and applications. The NetDAC COM object enables programmers to use ASP, PHP, and other server-side scripting languages to access Operational Insight data. You can also use NetDAC with Visual Basic or C++ to get data on the server. NetDAC cannot be used on the client side.

Except for methods used for connecting the server, NetDAC uses the same programming interface as the JWebDataSet. Programmers can then use the same skills, and the same code, when moving client-side applications to the server or vice versa.

**Note**
Unlike the JWebDataSet, NetDAC functions throw exceptions when an error occurs.

**Connecting to the NetServer**
You can connect to the Operational Insight server with NetDAC in one of two ways:
- Explicitly log on with a user name and password, creating a temporary session.
- Log on using an existing session.

**Explicitly Logging On**
You must use the `Login` function to log on to Operational Insight using a user name and password. The `Login` function takes the three parameters shown in the following table.
Parameters for Login Function

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UserName</td>
<td>A string containing the user name for Operational Insight log on.</td>
</tr>
<tr>
<td>Password</td>
<td>A string containing the Operational Insight password associated with the user name.</td>
</tr>
<tr>
<td>Host</td>
<td>A string containing the name of the computer that Operational Insight is running on. Can be an empty string if Operational Insight is running on the local machine.</td>
</tr>
</tbody>
</table>

Example: Using NetDAC in ASP with a new Operational Insight session.

```vbs<br%>
<%<br%>
dim ndDac<br%>
set ndDac = CreateObject("NetDAC.WebDataSet")<br%>
ndDac.Login "guestuser", "matrikon", ""<br%>
dim wrsResult<br%>
set wrsResult =<br%>
   ndDac.openRecordset("/ProcessNet/data/SQL_DS",<br%>
   "sql = select * from orders")<br%>
wrsResults.First<br%>
While wrsResult.Eof() = false<br%>
Response.Write wrsResult.Fields(0).AsString()<br%>
wrsResult.Next<br%>
Wend<br%>%>
```

Note

Sessions can be closed with NetDAC by using openRecordSet to call the closeSession RPC, as of Operational Insight 3.2.

Using an Existing Session in ASP

Example: Using NetDAC in ASP with an existing Operational Insight session.

```vbs<br%>
<%<br%>
dim ndDac<br%>
set ndDac = CreateObject("NetDAC.WebDataSet")<br%>
ndDac.FeedCookie unescape(Request.Cookies)<br%>
dim wrsResult<br%>
set wrsResult =<br%>
   ndDac.openRecordset("/ProcessNet/data/SQL_DS",<br%>
   "sql = select * from orders")<br%>
wrsResults.First<br%>
While wrsResult.Eof() = false<br%>
Response.Write wrsResult.Fields(0).AsString()<br%>
wrsResult.Next<br%>%>
```
The openSession function can also be used to connect to an existing session. It takes two numerical parameters: the session handle, and the key.

NetDAC RecordSet Object

The most common methods used with the record set are described in the following table.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetLastError</td>
<td>Returns a string representing any errors that occurred while opening the record set, or returns NULL if no errors occurred.</td>
</tr>
<tr>
<td>First</td>
<td>Moves the record pointer to the first record returned.</td>
</tr>
<tr>
<td>Next</td>
<td>Moves the record pointer to the next record.</td>
</tr>
<tr>
<td>Last</td>
<td>Moves the record pointer to the last record returned.</td>
</tr>
<tr>
<td>Prior</td>
<td>Moves the record pointer to the previous record.</td>
</tr>
<tr>
<td>Bof</td>
<td>Returns true if the record pointer is at the beginning of the results.</td>
</tr>
<tr>
<td>Eof</td>
<td>Returns true if the record pointer is at the end of the results.</td>
</tr>
<tr>
<td>Fields</td>
<td>Returns a reference to a specific field in the result set by numerical index.</td>
</tr>
<tr>
<td>FieldByName</td>
<td>Returns a reference to a specific field in the result set by field name.</td>
</tr>
<tr>
<td>FieldCount</td>
<td>Returns the number of fields in the result set.</td>
</tr>
<tr>
<td>RecNo</td>
<td>Returns the record number the record pointer is currently at.</td>
</tr>
<tr>
<td>RecordCount</td>
<td>Returns the number of rows in the result set.</td>
</tr>
<tr>
<td>ClearDataSet</td>
<td>Removes all the fields from the data set.</td>
</tr>
</tbody>
</table>

If you call Eof after calling Last, Eof will return true with the pointer on the last record.
If you call Eof after moving the pointer to the final record by calling Next, Eof will return false with the pointer on the last record.

### Writing BLOB Data Using NetDAC

NetDAC provides two methods for writing BLOB data into relational data sources, WriteStringBlob and WriteVariantBlob. The methods are similar; however, WriteVariantBlob accepts its BLOB data in a variant format.

#### Using WriteStringBlob

Using the WriteStringBlob method, applications can write string data into a BLOB field in a table. The values passed through the method’s parameters specify the data source to which you want to write, the SQL statement detailing where the BLOB data should be placed, and the string containing the BLOB data itself.

#### Parameters for WriteStringBlob Function

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>datasource</td>
<td>The name of the Operational Insight data source on which you want to run the query.</td>
<td>SQL_DS</td>
</tr>
<tr>
<td>SQL</td>
<td>The SQL statement describing where the BLOB data should be placed. A ? in the SQL statement operates as an alias syntax for the BLOB data itself.</td>
<td>sql = UPDATE mytable SET blobfield = ? WHERE someid = 123</td>
</tr>
<tr>
<td>BLOB</td>
<td>The string data to be placed in the BLOB field.</td>
<td></td>
</tr>
</tbody>
</table>

Example: Writing data in a BLOB field from an ASP page with an open Operational Insight session.

```vbscript
<% ' Create a NetDAC object  
    dim ndDac  
    set ndDac = CreateObject("NetDAC.WebDataSet")  
    ' Log into ProcessNet  
    ndDac.FeedCookie unescape(request.Cookies)  
    dim sBLOB, sQuery  
    sBLOB = "The Yellow Jack is yellow."  
    sQuery = "sql = UPDATE biolife set Notes =?  
      WHERE Common_Name = 'Yellow Jack'"
    ' Write the string data to the BLOB field.  
    ndDac.WriteStringBlob "SQL_DS", sQuery, sBLOB  
%>
```
Using WriteVariantBlob

The `WriteVariantBlob` method is similar to `WriteStringBlob`, except that its third parameter, the BLOB data, is required in a variant format. Programmers can use `WriteVariantBlob` to insert any kind of binary data into the BLOB field, instead of just simple text.

The third parameter in the `WriteVariantBlob` can be a variant containing either a string, or a variant array of bytes.

Example: Writing the contents of a binary file into a BLOB field using `WriteVariantBlob` in a VB application.

```vba
<%  
    Dim sQuery as String  
    Dim ndDac As New NetDac.WebDataSet  
    Dim vFileContents As Variant  
    ' Load in the file contents  
    Dim s As New Stream  
    s.Open  
    s.Type = adTypeBinary  
    s.LoadFromFile "C:\YellowJackNotes.doc"  
    s.Position = 0  
    vFileContents = s.Read  
    s.Close  
    ' Log into ProcessNet and write the contents  
    ' of the file into the BLOB field.  
    ndDac.Login "GuestUser", "matrikon", ""  
    sQuery = "sql = UPDATE biolife set Notes =?  
    WHERE Common_Name = 'Yellow Jack'"  
    ndDac.WriteVariantBlob "SQL_DS", sQuery, vFileContents  
%>
```

FMI

This example uses the ADO Stream object. For more information about using the ADO Stream object, see the MSDN online and search for “ado api reference”. For more information about the first two parameters (datasource and SQL), see “Using WriteStringBlob” on page 152.

Writing Tag Values Using NetDAC

In addition to retrieving data from Operational Insight, NetDAC also lets you write snapshot tag values into an historical data source, provided the Operational Insight driver supports this ability for that data source, and the tag is not read-only.

There are two methods for writing tag values, `WriteTag` and `WriteTags`. `WriteTag` provides the functionality to write a single tag’s value, and `WriteTags` the functionality to write multiple tag values in a single call.

Using WriteTag

Using the `WriteTag` method, you can write a single tag’s value and status. The `WriteTag` method has the parameters shown in the following table.
Parameters for WriteTag Function

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>datasource</td>
<td>A string containing the Operational Insight data source to which want to write the tag data.</td>
<td>OPC_SIM</td>
</tr>
<tr>
<td>tagname</td>
<td>A string containing the name of the tag for which you want to write values.</td>
<td>Random.int1</td>
</tr>
<tr>
<td>value</td>
<td>A string containing the value to be written.</td>
<td>12345</td>
</tr>
<tr>
<td>status</td>
<td>A long containing the OPC, or OPC-equivalent, status value to be written.</td>
<td>192</td>
</tr>
</tbody>
</table>

Example: Writing an OPC tag value using NetDAC in an ASP document, with an open Operational Insight session.

```<%'
    ' Create a NetDAC object
dim ndDac
    set ndDac = CreateObject("NetDAC.WebDataSet")
    'Log into ProcessNet
    ndDac.FeedCookie unescape(Request.Cookies)
    'Write the tag value dat
    ndDac.WriteTag "OPC_SIM", "Power.power1", "100", 192
%>```

Using WriteTags

The WriteTags method is similar to the WriteTag method, except that the three parameters that specify the tag name, value, and status have been replaced with arrays of variants containing tag names, values, and statuses. Each of these arrays must be of equal length, meaning that there may only be one value and one status for each tag.

Parameters for WriteTags Function

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>datasource</td>
<td>A string containing the Operational Insight data source to which you want to write the tag data.</td>
<td>OPC_SIM</td>
</tr>
<tr>
<td>tagnames</td>
<td>A variant array containing tag names.</td>
<td></td>
</tr>
</tbody>
</table>
Example: Writing multiple tag value and statuses using the `WriteTags` method, in an ASP page with an open Operational Insight session.

```vbscript
<%  
    Dim aTagNames(2)  
    Dim aTagValues(2)  
    Dim aTagStatuses(2)  
    Dim ndDac  

    ' Create a NetDAC object  
    dim ndDac  
    set ndDac = CreateObject("NetDAC.WebDataSet")  

    ' Log into ProcessNet  
    ndDac.FeedCookie unescape(Request.Cookies)  

    ' Set up the arrays containing the tag values  
    ' and statuses to be written  
    aTagNames(0) = "Power.power1"  
    aTagValues(0) = "150"  
    aTagStatuses(0) = 192  

    aTagNames(1) = "Power.power6"  
    aTagValues(1) = "25"  
    aTagStatuses(1) = 192  

    aTagNames(2) = "Power.power3"  
    aTagValues(2) = "1.5"  
    aTagStatuses(2) = 192  

    ' Write the tag value data  
    ndDac.WriteTags "OPC_SIM", aTagNames, aTagValues, a TagStatuses  
%>
```

**Parameter** | **Description** | **Example**
--- | --- | ---
values | A variant array of string values, one for each tag in the `tagnames` array. |  
statuses | An array of longs containing the OPC, or OPC-equivalent, status values to be written, where there is one status array element for each element in the `tagnames` array. |  

FMI

The previous sections describe only portions of the programming interface. For more information, refer to the JavaDocs, located in the javadocs folder on the installation CD.
Chapter 12: Using ADO to Get Operational Insight Data

You can specify the ADO format when performing URL queries to retrieve data in an XML format. This format is named ADO because it uses a schema compatible with Microsoft ADO (ActiveX Data Objects). ADO is often used with Visual Basic and other languages to get access to data in databases. You can also use ADO to request data from Operational Insight for use in custom applications.

Excel and ADO

Microsoft Excel provides a VBA development environment that you can use to create ADO result sets. You can use these result sets to get access to data from Operational Insight. The only way you can use the ADO packet provider is to retrieve data through HTTP, just as you would in the Web browser.

Using a Recordset Object

To use ADO within Excel, you must reference the appropriate type library.

To use ADO in Excel

1. Start Excel, and open the VBA editor by pressing ALT-F11.
2. In the VBA environment Tools menu, click References.
3. From the list of available libraries, select the latest version of Microsoft ActiveX Data Objects library.
4. Create and use a record set object to access Operational Insight data, as shown in the following example.

Example: Creating a record set.

Sub GetData()
    Dim rs As New Recordset
    rs.Open = "http://[svr]/[path]/netportal.dll/data/opc_sim?" + _
        "tag0 = Random.Int2&start = *-10M&" + _
        "end = *&function = interp&step = 1<&format = ado&" + _
        "username = guestuser&password = matrikon"
    MsgBox rs.RecordCount
End Sub

The previous example makes a request to the server, and then displays the number of records returned. In this example, the user name and password are passed to the server along with the request. This is required the first time the GetData function is invoked. Subsequent calls do not need to pass the user name or password, although doing so would not cause problems. Either way, it is best not to hard code passwords; it would be better for the user to enter the password from a dialog box.
The request made in the previous example retrieves tag values, but you can modify it to execute a SQL statement instead. All URL queries that are valid in the Web browser also work with ADO.

Example: Populating a worksheet with data.

```vba
Sub GetData()
    Dim rs As New Recordset
    rs.Open = "http://[svr]/[path]/netportal.dll/data/opc_sim?" + _
    "tag0 = Random.Int2&start = *-10M" + _
    "end = *&function = interp&step = 1&lt;format = ado" + _
    "username = guestuser&ampassword = matrikon"
    Dim fld As Integer, row As Integer
    Dim fldcount As Integer
    fldcount = rs.FieldsCount
    row = 6
    While rs.EOF = False
        For fld = 1 To fldcount
            Cells(row, fld) = rs.Fields(fld - 1).Value
        Next
        rs.MoveNext
        row = row + 1
    Wend
End Sub
```

Using MSXML

The major drawback to using record set objects for data retrieval is that they do not report useful error information. If a record set's data request fails, it generates an error, but does not provide a description of why it failed. One way to work around this limitation is to use the MSXML object. MSXML contains a lot of functionality, including XML parsing and data requests through HTTP. You can use MSXML to perform the data request, then you can manually check the returned data for errors, and finally initialize the record set object with the stream provided by the MSXML object.

Example: Using the MSXML object in VB to access Operational Insight data and display errors if they occur. For this example to work, the VB/VBA project must reference the Microsoft XML V3.0 and ActiveX Data Objects libraries.

```vba
Dim xReq As New XMLHTTP30
xReq.open = "POST"
"http://[svr]/[path]/netportal.dll/data/opc_sim?tag0 = Random.Int2
&format = ado&username = guestuser
&ampassword = matrikon", False
xReq.send
If xReq.responseXML.xml = "" Then
    MsgBox "error:" & xReq.responseText
Else
    Dim rs As New Recordset
    rs.open xReq.responseStream
    MsgBox rs.Fields.Count
End If
```
Chapter 13: Creating Packet Providers

You can create a packet provider by creating a COM object using Visual Basic, C++, or Delphi. When a data request is processed by the NetPortal, it checks to see which presentation format has been requested, using an HTTP query parameter called Format. If the format is not an internal format, the NetPortal attempts to load a packet provider for the requested format. If a suitable provider is found, the data set is sent to it for formatting, and the provider returns either a text string containing the formatted data, or a binary array of bytes.

Obtaining a Provider

NetPortal uses the value of the Format query parameter to format a data result set (or data set). The following table lists how the packet provider is acquired.

Packet Provider Formats

<table>
<thead>
<tr>
<th>Format String</th>
<th>Provider Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;blank&gt; or NST</td>
<td>A blank format specification or NST causes the NetPortal to use the internal binary provider for the JWebDataSet.</td>
</tr>
<tr>
<td>HTML</td>
<td>The data is formatted as an HTML table using the internal HTML provider.</td>
</tr>
<tr>
<td>Excel</td>
<td>The data is formatted as a tab- and line-feed separated string table using the internal Excel provider.</td>
</tr>
<tr>
<td>CSV</td>
<td>The data is formatted as a comma-separated string table using the internal CSV provider.</td>
</tr>
<tr>
<td>other</td>
<td>When none of the internal providers is specified, the NetPortal attempts to load an external provider using the provider’s COM ProgID. This ProgID is created when the packet provider is compiled and registered. The NetPortal appends the format string to a PacketProvider in order to generate the full provider name. For example, if the format string was MyXML, the NetPortal would attempt to load a COM object with the ProgID of PacketProvider.MyXML. If this provider is not found, an error message is sent to the user.</td>
</tr>
</tbody>
</table>

Packet Provider Interface

Packet providers must implement the IProNetPacketProvider interface. This is described in the type library PNPacketProvider.tlb, which contains the description of PNPacket Provider Type Library. The following table lists the interface methods in the packet provider interface.
Note

The PNPacketProvider.tlb file is not part of the default Operational Insight installation. Please contact Product Support at productsupport@honeywell.com to receive a copy of this file.

Packet Provider Interface Methods and Properties

<table>
<thead>
<tr>
<th>Method/Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IsBinary (property)</td>
<td>Returns false if the GetFormattedPacket method returns a text string, and true if it returns an array of bytes.</td>
</tr>
<tr>
<td>ContentType (property)</td>
<td>Returns the mime type of the format.</td>
</tr>
<tr>
<td>GetFormattedPacket (method)</td>
<td>Returns the formatted data.</td>
</tr>
</tbody>
</table>

Method Details

This section describes the parameters and return types for each of the interface methods.

Packet Provider Interface Method Details

<table>
<thead>
<tr>
<th>Method/Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IsBinary (property)</td>
<td>A Boolean property that indicates whether packet is binary.</td>
</tr>
<tr>
<td>ContentType (property)</td>
<td>A string property that returns the mime type of the packet. If an empty string is returned, the NetPortal defaults to text/html.</td>
</tr>
<tr>
<td>GetFormattedPacket (method)</td>
<td>A method that formats the NetPortal data set, and returns it to the NetPortal for delivery to the client.</td>
</tr>
</tbody>
</table>
The parameters used by `GetFormattedPacket` are shown in the following table.

### Parameters for `GetFormattedPacket` Function

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WdsData</td>
<td>[in] VARIANT</td>
<td>A variant array containing the NetPortal data set, in Row, Column format. The first row contains the field names.</td>
</tr>
<tr>
<td>sHTTPQuery</td>
<td>[in] BSTR (String)</td>
<td>A comma-separated list of the query parameters as passed to the NetPortal.</td>
</tr>
<tr>
<td>sHTTPQuery</td>
<td>[out, retval] VARIANT*</td>
<td>The returned data, formatted. This variant contains either a BSTR or an array of bytes if IsBinary returns true.</td>
</tr>
</tbody>
</table>

### Sample Packet Provider

You can create a custom packet provider using VB, C++, Delphi, or other language that supports COM. The following descriptions and code illustrate a simple packet provider created in Visual Basic.

#### Creating the Project

To create a packet provider in VB, you must create the VB project, reference the PNPacketProvider library, and implement the interface.

#### To create a packet provider in Visual Basic

1. Start VB, and when prompted, choose to create an **ActiveX DLL** project.
2. Change the name of the default class from `Class1` to the name of the format you want to create. For example, change `Class1` to `MyTable`.
3. Click on the root project item in the project viewer to view the project name in the **Object Inspector**.
4. Change the project name from `Project1` to `PacketProvider`.
5. On the menu, point to **Project** and click **References** to display a list of available objects and interfaces.
6. In the list, select the **PNPacketProviderTypeLibrary** and make sure the check box is selected.
7. In the **MyTable** class window, type the following:
   ```vbnet
   implements PNPacketProvider.IProNetPacketProvider
   ```
In the list at the top left of the editor, choose IPNPacketProvider. The right drop-down list now displays the methods of the interface that you must implement.

Implement each of these methods by selecting it from the right-hand drop-down list and entering the code. See “Sample Code” on page 162 for an example.

Save the project using a suitable name, such as MyTableProvider.vbp.

Compile the project.

Test the provider by entering a NetPortal data query in the Web browser, and specifying your new provider. For example:

```
http://[svr]/[path]/netportal.dll/data/SQL_DS?sql = select * from industry&format = MyTable
```

where [svr] is the computer name where Operational Insight is running, and [path] is the server's path to NetPortal, which by default is: `<ProcessNet/isapi/netportal/`.

### Sample Code

The following code was generated using the procedure described above, and adding format-specific code to the methods:

```vba
Implements PNPacketProvider.IProNetPacketProvider

Private Property Get IProNetPacketProvider_IsBinary() As Boolean
    IProNetPacketProvider_IsBinary = False
End Property

Private Function IProNetPacketProvider_GetFormattedPacked(ByVal saWdsData As Variant, ByVal sHTTP Query as String) As Variant
    Dim sResponse As String
    Dim nField As Integer, nNumFields As Integer
    Dim nRow As Integer, nNumRows As Integer
    ' variant array is the result set in row, column format,
    ' with the first row being the field names
    nNumRows = UBound(saWdsData, 1)
    nNumFields = UBound(saWdsData, 2)
    ' build simple table for response
    sResponse = "<html><body>This is my table format"
    sResponse = sResponse + "<br><table border = 0 bgcolor = silber><tr>"
    ' first table header (field names)
    For nField = 1 To nNumFields
        sResponse = sResponse + "<td>" & saWdsData(nRow, nField) & "</td>"
    Next
    ' build simple table for response
    sResponse = sResponse + "</tr></table>""
```
sResponse = sResponse + "</tr></table></body></html>"
' return the content
IProNetPacketProvider_GetFormattedPacket = sResponse
End Function
Chapter 14: NetDraw Software Development Kit

Programmers can use the NetDraw Software Development Kit (SDK) to customize NetDraw. The NetDraw SDK exposes functionality to create custom objects, and configurable design mode and visibility.

NetDraw GUI Configuration

There are two components you can customize in the NetDraw interface:

- Defining custom toolbars
- Defining buttons

NetDraw uses a master configuration file for configuration. By default, NetDraw uses this master configuration file:

```
config/default/defaultmaster.xml
```

You can create other master configuration files by referring to the masterconfig.dtd file.

**Note**

The default XML files for NetDraw (located in config/default) provide a good example of how to write a configuration file set. However, it is recommended that you leave these files intact and create another set of configuration files in an appropriately named directory. You can copy the config/default folder and rename the folder and the master configuration file, before changing the configuration files.

Defining a NetDraw Configuration File

To work with the configuration files, you must have a working knowledge of XML. In order to create non-visual custom objects, you should have a working knowledge of HTCs.

The configuration file DTD contains a detailed usage description for each element and property.

You can use a master configuration file to specify other configuration files, and define key/value pairs. These key/value pairs can be accessed through script in design mode.

There are several optional system key/value pairs that NetDraw uses internally, as shown in the following table.
### NetDraw System Key/Value Pairs

<table>
<thead>
<tr>
<th>Key</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ShowPropertyInspector</td>
<td>false</td>
<td>Determines whether the Object/Property Inspector is displayed.</td>
</tr>
<tr>
<td>ShowSymbolLibrary</td>
<td>false</td>
<td>Determines whether the Symbol Library is displayed.</td>
</tr>
<tr>
<td>ShowTopToolbar</td>
<td>false</td>
<td>Determines whether the top toolbar is displayed.</td>
</tr>
<tr>
<td>bAutoCompletePoly</td>
<td>true</td>
<td>Determines whether polygon objects are auto-completed.</td>
</tr>
<tr>
<td>bTransparency</td>
<td>true</td>
<td>Determines whether NetDraw objects are transparent during move/resize.</td>
</tr>
<tr>
<td>iNumUndoLevels</td>
<td>5</td>
<td>Specifies the number of undo levels.</td>
</tr>
<tr>
<td>iGridSizePref</td>
<td>15</td>
<td>Specifies the grid spacing in pixels (0=Off).</td>
</tr>
<tr>
<td>bAutoSelect</td>
<td>true</td>
<td>Determines whether objects are auto-selected.</td>
</tr>
<tr>
<td>bEdgeDetect</td>
<td>true</td>
<td>Determines whether the top/left screen edges are detected.</td>
</tr>
<tr>
<td>bButtonText</td>
<td>true</td>
<td>Determines whether the text under toolbar buttons is shown.</td>
</tr>
<tr>
<td>sBkColor</td>
<td>silver</td>
<td>Specifies the background color.</td>
</tr>
<tr>
<td>sLanguage</td>
<td>vbscript</td>
<td>Specifies the scripting language (vbscript or jscript).</td>
</tr>
<tr>
<td>sLineColor</td>
<td>black</td>
<td>Specifies the line color.</td>
</tr>
<tr>
<td>sFillColor</td>
<td>white</td>
<td>Specifies the fill color.</td>
</tr>
<tr>
<td>sTextColor</td>
<td>black</td>
<td>Specifies the text color.</td>
</tr>
<tr>
<td>sFont</td>
<td>Arial</td>
<td>Specifies the font.</td>
</tr>
</tbody>
</table>
Loading a NetDraw Configuration

To load an alternative NetDraw configuration, the master configuration file must be specified in the NetDraw URL using the following format:

```
http://localhost/ProcessNet/isapi/netportal/netportal.dll/html/NetDraw.html?MasterConfig=%configxml%
```

<table>
<thead>
<tr>
<th>Key</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sSize</td>
<td>10</td>
<td>Specifies the font size, in points.</td>
</tr>
<tr>
<td>bBold</td>
<td>false</td>
<td>Determines whether the font is bold.</td>
</tr>
<tr>
<td>bItalics</td>
<td>false</td>
<td>Determines whether the font is italicized.</td>
</tr>
<tr>
<td>bUnderline</td>
<td>false</td>
<td>Determines whether the text is underlined.</td>
</tr>
<tr>
<td>sAlign</td>
<td>left</td>
<td>Specifies how the text is aligned.</td>
</tr>
<tr>
<td>bAutoSave</td>
<td>true</td>
<td>Determines whether the page will be auto-saved before it is previewed.</td>
</tr>
<tr>
<td>bAutoSavePrompt</td>
<td>true</td>
<td>Determines whether users will be prompted before the page is auto-saved.</td>
</tr>
<tr>
<td>gsRowX</td>
<td>215</td>
<td>Specifies the width of the Object/Property Inspector frame.</td>
</tr>
<tr>
<td>bShowInspector</td>
<td>false</td>
<td>Determines whether the Object/Property Inspector frame is shown on load.</td>
</tr>
<tr>
<td>ToolType</td>
<td>NDRAW</td>
<td>Specifies the tool type you want to use for content loading/saving.</td>
</tr>
<tr>
<td>Normalize</td>
<td>doLoadFromHTML(aReturn)</td>
<td>Specifies the function you want to call to normalize design mode.</td>
</tr>
<tr>
<td>Serialize</td>
<td>getPageHTML(null, true)</td>
<td>Specifies the function you want to call to retrieve savable content.</td>
</tr>
</tbody>
</table>
Where `%configxml%` is the file path to the master configuration file.
For example:
```
```

**Defining Custom Toolbars and Buttons**

Programmers can use the NetDraw SDK to customize both the Top and Symbol Library coolbars (toolbars that contain, manage, and display child windows; such as other toolbars, combo boxes, and bitmaps). The floating toolbars are also completely customizable, and can be created, altered, or removed by editing an XML file; you do not need to write any code.

You can define additional coolbars, although it is up to the developer to implement the actual toolbar.

---

**Note**

If you remove a button corresponding to a design mode context menu item, the shortcut menu item is also removed. This means that if you remove the Cut button from the Top toolbar, the Cut option will no longer be available when right-clicking an object.

---

**Defining Custom Buttons**

All buttons used within the application are defined in an XML file called `buttonconfig.xml`, which is specified (via an include statement) in the master configuration file. This file defines the image source and size, popup text, label, ID, and event handlers.

You can use buttons that are 24 or 16 pixels in size. However, to maintain a consistent look, all buttons for a specific toolbar should be the same size. Buttons for coolbars should be 16 pixels in size.

There are two event handlers that you can define for a button: `onbuttonclick` and `ondblclick`. The function you specify to be called when these events occur will receive the `buttonID` of the ToolbarButton (if specified in `toolbarchconfig.xml`) as its only parameter.

**Example:** The button definition and event handler definition.
```
<Button buttonID="alignleft" popup="^Align left edges^">
  <Image src="/ProcessNet/netdraw/img/align/alignleft.gif" size="24">
  </Image>
  <EventHandler Handler="alignleft"></EventHandler>
</Button>

function alignleft(sButtonID)
{
  doMoveSelTo("left");
} //alignleft
```
All of the buttons on the same coolbar can specify the same event handler. The event handler can then use the buttonID parameter in a switch statement.

Defining Custom Toolbars
All of the coolbars and floating toolbars are defined in an XML file called toolbarconfig.xml, which is specified (via an include statement) in the master configuration file. For coolbars, you can specify the frame name, coolbar name, and whether to show text below coolbar buttons. For floating toolbars, you can specify the toolbar title, ID, default positioning, default visibility, default orientation, and whether to use user-defined breaks.

Each coolbar or floating toolbar definition contains a list of buttons to display, as well as the default background color and caption color. These buttons must be referenced from the button configuration file (buttonconfig.xml).

You can specify additional toolbar button parameters which control button features such as toggling, border visibility, and shortcut key. The logic to be used for determining when to enable or disable the button can also be specified. For custom rules, the specified function receives an array of selected objects as its only parameter.

Useful Functions
See script/systemtoolbars.js and script/custombuttonlogic.js for sample implementations of some of the following functions.

Note
For a listing of object types, refer to the title of the shape elements in inspector/PropertyInspector.xml.

Returns true if none of the object types specified in aTypes are within aSelection:

```javascript
isNoneInSelection(aSelection, aTypes)
```

Returns true if any of the object types specified in aTypes are within aSelection:

```javascript
isAnyInSelection(aSelection, aTypes)
```

Returns true if all of the object types specified in aTypes are within aSelection:

```javascript
isAllInSelection(aSelection, aTypes)
```

Returns true if only the object types specified in aTypes are within aSelection:

```javascript
isOnlyInSelection(aSelection, aTypes)
```

Toggles the visibility of the specified floating toolbar:

```javascript
toggleToolbar(sFloatingToolbarID)
```
Chapter 14: NetDraw Software Development Kit

Examples

This section contains some examples of NetDraw GUI configuration, and advanced details about the implementation of a custom configuration file.

- “Accessing Master Config Key/Value Pairs” on page 170
- “Writing a Custom Button: EnableLogic Function” on page 170
- “Writing a Button Event Handler” on page 171
- “Advanced: Including and Implementing a Custom Configuration File” on page 171

Accessing Master Config Key/Value Pairs

The Master Configuration File object can only be accessed from design mode.

```javascript
var mapConfigData = parent.goMasterConfig.mapConfigData,
    oValue = mapConfigData.item("ShowTopToolbar");

if (typeof(oValue) != "undefined")
    alert(oValue);
else
    alert("Could not find key.");
```

Writing a Custom Button: EnableLogic Function

If the EnableLogic function for a toolbar button in toolbarconfig.xml contains a CustomRule element with a doMyButtonLogic function, the definition of this function could be any one of the following:

```javascript
function doMyButtonLogic(aSelection)
{
    return(false); // Always disable the button
} //doMyButtonLogic

function doMyButtonLogic(aSelection)
{
    // Only enable the button if five rectangle objects are selected
    var bFive = (aSelection.length == 5),
        bOnlyRects = isOnlyInSelection(aSelection,
            new Array("rectangle");
    return(bFive && bOnlyRects);
} //doMyButtonLogic

function doMyButtonLogic(aSelection)
{
    // Enable the button if EnableMyButton is set to T in the master config file
    var mapConfigData = parent.goMasterConfig.mapConfigData,
        oValue = mapConfigData.item("EnableMyButton");

    return((typeof(oValue) != "undefined") && (oValue == "T"));
} //if} //doMyButtonLogic
```
Writing a Button Event Handler

There are two approaches to handling button events. The first approach is to define a separate function for each event. If there are two buttons defined in buttonconfig.xml with the event handlers test and example, two functions could be created as follows:

```javascript
function test(sButtonID)
{
    alert("test");
} //test

function example(sButtonID)
{
    alert("example");
} //example
```

The second approach is to define a single function per toolbar, and use the buttonID for selection in a switch statement. This requires that a unique buttonID is defined for each ToolbarButton in toolbarconfig.xml. The function could look like the following:

```javascript
function doToolbarButtonEvent(sButtonID)
{
    switch(sButtonID)
    {
    case "test": alert("test"); break;
    case "example": alert("example"); break;
    default: alert("unhandled button event");
    } //switch
} //doToolbarButtonEvent
```

Advanced: Including and Implementing a Custom Configuration File

You can use the master configuration file to define custom key/value pairs, which can be used by custom code in design mode. If you want to store more information than the key/value pairs store, you can create a custom configuration file, and include it in the master configuration file. The master configuration loader will parse this custom configuration file in addition to the standard configuration files, and make the object model in the custom configuration file available to design mode code.
You can place the custom configuration file’s DTD in the `config/DTD` directory, and the custom configuration file itself can be placed along with the other configuration files in `config/nameofapp`.

In order for the content in the custom configuration file to be available in design mode, you must create a JScript object with a constructor that has the same name as the `name` attribute on the ConfigFile element in the master config file. This JScript object must have a method called “normalize” that receives an `xmlDataIsland` as its only parameter. The JScript object must provide member properties or methods as needed in order to contain the data defined in the custom configuration file. The normalize method populates these member properties or methods from the XML.

The JScript object is accessed from design mode as follows:

```
parent.goMasterConfig.mapConfigObjects.item(sConfigFileName)
```

where `sConfigFileName` is the value of the `name` attribute on the ConfigFile element in the master configuration file, for example, `toolbarconfig.xml`.

---

**FMI**

The JScript object and supporting functions can be placed in a custom script file and included in NetDraw. For more information, see “Including Script in NetDraw” on page 172. The standard NetDraw configuration files have JScript objects defined in `script/configobjects.js`, which can be used as an example.

---

**Including Script in NetDraw**

Although you can modify the actual NetDraw files with custom script, it is not recommended. Operational Insight upgrades can overwrite these files, and remove the custom changes. You should store all configuration files and code in a subfolder of the folder `/config/`.

For example, a customized version of NetDraw report builder might have the following directory structure:

- `/config/reportbuilder/` This folder contains the six required configuration files (including the master configuration file).
- `/config/script/` This folder contains any custom script files required for the customized NetDraw, including event handlers, button logic, and custom object definitions.
- `/config/img/` This folder contains the images used for any custom buttons displayed on toolbars.

There are several advantages to keeping all of the files for a custom NetDraw configuration in one place:

- Simplified management and back up.
- Easy replication to other PN servers (copy over a single folder).
- Less risk of losing customizations during an upgrade or patch (better forward compatibility).
Property Inspector

All properties displayed in the **Property Inspector** are defined in `config/default/propertyinspectorconfig.xml`. Each property is defined using a `<Property>` element, and you can define attributes such as the default value, edit box or drop-down list, getter/setter code, and validation codes.

**Working with Properties**

To display a property as a child of another property, set the `parentTitle` attribute to the title of the parent property. Define the parent property before its children in the configuration file.

**Getter/Setter Code**

For each property, you must specify what to display as its value. This value typically corresponds to an attribute on a NetDraw object. For simple properties, such as `Width`, you can enter templated JScript code to define how to retrieve the value into the `<ValueGetter>` element, for example:

```xml
<ValueGetter>
  [#object].style.pixelWidth
</ValueGetter>
```

You must also specify how the property will apply a new value to an object. When the user enters a new value, by typing in a new value, selecting a new value from a drop-down list, or by using a property editor, the `<ValueSetter>` element defines how to apply this new value to an object. If the property refers directly to an object property, the same code can be used for both the getter and setter. For example, the `Width` property directly corresponds to the `style.pixelWidth` object property, and because of this, both the getter and setter code for the `Width` property are the same.

Both the getter and setter contain templated code, which means that certain text has special meaning. A list of the items that can be used in getter/setter code is shown in the following table.

**Getter/Setter Code Items**

<table>
<thead>
<tr>
<th>Text</th>
<th>Represents</th>
</tr>
</thead>
<tbody>
<tr>
<td>[#object] or oObj</td>
<td>The actual NetDraw object.</td>
</tr>
<tr>
<td>oProperty</td>
<td>An instance of a Property object that corresponds to the applicable property.</td>
</tr>
<tr>
<td>sNewValue</td>
<td>(Setter only) A text representation of the value for the <strong>Property Inspector</strong> property.</td>
</tr>
<tr>
<td>sOldValue</td>
<td>(Setter only) A text representation of the previous value for the property.</td>
</tr>
<tr>
<td>oData</td>
<td>(Setter only) A generic object that contains any extra data returned from a property editor.</td>
</tr>
</tbody>
</table>
For properties that do not actually represent an actual object property, the getter/setter code may not be the same. For example, the ColorAnim property is a complex property that represents configuration information for an animation. The getter for this property references an external function that determines what the value for this property is for a given object. The setter for this property references a different external function that applies the configuration data to an object.

Validating a Property Value

Validation can be performed when a user enters a new value for a property. A property can make use of predefined validation criteria, or it can reference an external function to perform the validation.

The predefined validation criteria consist of one of four data types, and an option range for numerical data types. If you specify a data type for a property, when the user enters a new value for the property, it is validated to ensure that it is of that data type. For example, if the Width property has a data type of int, the new value must be a whole number. If the Width property also specifies a range of 0,22766, the whole number must be greater than or equal to 0, and less than or equal to 22766.

You can call an external function to perform validation by specifying the custom attribute on the <Validation> element. This attribute is used for validation requirements that cannot be satisfied by using the predefined validation criteria. The function specified by the custom attribute receives two parameters, oProperty (an instance of a Property object that corresponds to the applicable property), and sNewValue, which is the value to which the Property Inspector row will be set if validation is successful. The function indicates successful validation by returning true; returning false will cancel the action.

Defining the Objects that can Use a Property

All NetDraw objects, including custom objects, need to specify which properties (if any) to show in the Property Inspector when they are selected. A NetDraw object is represented in the configuration file using the <Shape> element. The <Shape> element can either directly reference properties, or it can reference a group of properties.

You can group common or closely related properties, so that each applicable <Shape> can reference the group of properties. Groups of properties are represented with the <PropertyGroup> element. Both the <Shape> and <PropertyGroup> elements can contain <TabContent> elements. The <TabContent> element specifies which properties to display under a specific tab, where 1 represents the first tab.

Working with Property Editors

A property editor represents a custom data input method, and is shown in the Property Inspector as an ellipsis button. Property editors can be used when a standard text box or drop-down list box cannot be used to define a value for a property. Property editors can also serve as helpers for text input.
For example, a SQL property could have a text box in which the user could manually enter a query, but the property could also have a property editor that would launch NetQuery and allow the user to graphically define a query.

Each property editor is defined at the beginning of the configuration file using the `<PropertyEditor>` element. You can use the `<PropertyEditor>` element to specify an external function that is called when a user opens the property editor using the ellipsis button. Two parameters are passed to the external function: `oProperty` (an instance of a Property object that corresponds to the applicable property), and `oRow` (an instance of a Row object that represents a row in the Property Inspector). The external function must return an array of three elements as follows:

- [0]: A Boolean, set to true if the property editor is cancelled.
- [1]: A generic object (oData), available in the setter code.
- [2]: A string that represents the new value of the Property Inspector row.

For example, the system property editors all point to the `doSystemPropertyEditor()` function (located in script/propertyeditors.js), which looks at `oProperty.Custom.ID` to determine which property editor function to call.

```javascript
// custom row object, represents a row on a tab
function Row(sTitle, sValue, sType, aValues, bReadOnly, bEllipses, oParentRow, iMaxLength)
{
    // properties
    this.title = sTitle;
    this.value = sValue;
    this.type = sType;
    this.values = aValues;
    this.readonly = bReadOnly;
    this.ellipses = bEllipses;
    this.parentRow = oParentRow;
    this.maxlength = iMaxLength;

    this.expanded = false;
    this.selected = false;

    this.id_A = getUniqueRowID();
    this.id_B = getUniqueRowID();
    this.id_childA;
    this.id_childB;
    this.id_value = getUniqueRowID();
    this.aChildren = new Array();

    // methods
    this.addChild = Row_addChild;
    this.getRowHTML = Row_getRowHTML;
    this.isUniqueTitle = Row_isUniqueTitle;
    this.getRowByID = Row_getRowByID;
    this.getControlObj = Row_getControlObj;
    this.setControlVisible = Row_setControlVisible;
    this.getRowByTitle = Row_getRowByTitle;
    this.getAllRows = Row_getAllRows;
}; //Row_constructor
```
Note
Do not alter existing property editors (for example, `doSystemColorEditor`). Instead, copy and paste the property editor into a new file, edit the new file, and then point the case statement to the new function.

Custom Objects

In NetDraw, a custom object is any HTML object that is compatible with NetDraw. Custom objects are always inserted into NetDraw, as opposed to being drawn. For example, a custom object can be inserted into NetDraw the same way as a text object, but not the same way as a rectangle, whose boundaries are defined before the object is created.

Creating a Custom Object

You can use the configuration file `config/default/customobjectsconfig.xml` as an example of creating custom objects. The XML conforms to the DTD `config/dtd/customobjectsconfig.dtd`. This DTD documents each element and attribute that is available for defining a custom object.

Guidelines for Defining a Custom Object

To create and use a custom object, you must ensure that the object definition is compatible with NetDraw. The definition of a custom object is the HTML inserted into the page when an object is created. This HTML is generated when the code in the object definition is executed. In the following example definition, the function `getTextHTML()` was written and placed in `script/customobjects.js`:

```xml
<Definition>getTextHTML();</Definition>
```

The custom function `getTextHTML()` returns an array of two elements (or `null`/`false` to indicate the cancellation of the operation). The first element is a string representing the HTML to be inserted. The second element is the ID of the object. The second element is typically `null`, because the ID is usually auto-generated using a token to represent the ID in the HTML. However, if the custom function needs its own ID for the object, it must be returned as the second element of the array.

Several tokens that are automatically replaced with the appropriate value are shown in the following table.
Auto-Generated Tokens

<table>
<thead>
<tr>
<th>Token</th>
<th>Example Usage</th>
<th>Gets replaced with</th>
</tr>
</thead>
<tbody>
<tr>
<td>%ID%</td>
<td>id=&quot;%ID%&quot;</td>
<td>An auto-generated ID based on the object name.</td>
</tr>
<tr>
<td>%WIDTH%</td>
<td>style=&quot;width:%WIDTH%&quot;</td>
<td>The default object width.</td>
</tr>
<tr>
<td>%HEIGHT%</td>
<td>style=&quot;height:%HEIGHT%&quot;</td>
<td>The default object height.</td>
</tr>
<tr>
<td>%TOP%</td>
<td>style=&quot;top:%TOP%&quot;</td>
<td>The default object y-coordinate.</td>
</tr>
<tr>
<td>%LEFT%</td>
<td>style=&quot;left:%LEFT%&quot;</td>
<td>The default object x-coordinate.</td>
</tr>
<tr>
<td>%ZINDEX%</td>
<td>style=&quot;z-index:%ZINDEX%&quot;</td>
<td>An auto-generated z-index. We recommend using this token; otherwise problems with object layering and the ordering tools can result.</td>
</tr>
<tr>
<td>%OBJECTTYPE%</td>
<td>ObjectType=&quot;%OBJECTTYPE%&quot;</td>
<td>An auto-generated object type consisting of the word “insert” plus the object name. We recommend using this token, because an object with an invalid ObjectType could cause application errors.</td>
</tr>
<tr>
<td>%NAME%</td>
<td>title=&quot;%NAME%&quot;</td>
<td>The name of the object as defined in the configuration file.</td>
</tr>
<tr>
<td>%NONVISUAL_ICON_SRC%</td>
<td>title=&quot;%NONVISUAL_ICON_SRC%&quot;</td>
<td>The path specified for srcIcon in the object definition. This only applies to non-visual objects, and is used internally.</td>
</tr>
<tr>
<td>%NONVISUAL_HTC_SRC%</td>
<td>title=&quot;%NONVISUAL_HTC_SRC%&quot;</td>
<td>The path specified for srcHTC in the object definition. This only applies to non-visual objects, and is used internally.</td>
</tr>
</tbody>
</table>

The returned HTML must either be a VML element or be contained in a VML element for it to be compatible with NetDraw. If a non-VML object is defined, it
must be wrapped in a VML element, such as the rectangle \(<\texttt{v:rect} />\). Many of the object HTML definitions in script/customobjects.js are defined in this way.

All tokens (except the ones that use the title attribute as the example usage) must be defined on the topmost element. This means that if you want to define a \(<\texttt{textarea}>\) element as a custom object, it must be wrapped in a VML rectangle element, and the tokens must be defined on the rectangle, not the text area. The text area’s width and height should be 100 percent if you want the text area to match the width and height of the parent element (rectangle) as it is resized.

Only box objects can be defined as custom objects. A box object is an object with width and height; for example, a rectangle or an image. A line is an example of an object that is not a box object.

Some properties available for an object in the Property Inspector rely on the presence of certain VML children in a custom object definition. These child objects should be the first children of the custom object. The following table contains the Property Inspector property and corresponding required child object.

### Child Objects Required by Property Inspector Objects

<table>
<thead>
<tr>
<th>Property</th>
<th>Required Child Object</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stroked</td>
<td>(&lt;\texttt{v:stroke} /&gt;)</td>
<td>Makes a border property available for the object.</td>
</tr>
<tr>
<td>Shadow</td>
<td>(&lt;\texttt{v:shadow} /&gt;)</td>
<td>Makes a shadow property available for the object, with which a shadow can be placed behind an object.</td>
</tr>
<tr>
<td>Filled</td>
<td>(&lt;\texttt{v:fill} /&gt;)</td>
<td>Makes a fill property available for the object.</td>
</tr>
</tbody>
</table>

Keep these guidelines in mind when defining a custom object:

- Ensure that the returned HTML is based on a VML object or container object.
- Ensure that a property return value is used by the custom function (null/false for cancel).
- Use the special tokens whenever possible.
- Ensure that the proper VML children exist if the object will be exposing the equivalent properties through the Property Inspector.

### Inserting a Custom Object Into the Page

Once you have defined a custom object in the configuration file, the next step is to expose it in design mode so that it can be used. This can be done in one of two ways: by defining a button to insert the object, or by manually inserting the object through JScript.

The function called when the button is clicked can insert a custom object by setting NetDraw to the appropriate mode. For example, to cause a Text object to be inserted, the following line of code must be executed:
document.body.setMode("inserttext");

This is the same code that must be executed if you want to insert an object into a NetDraw page manually, without clicking a button.

FMI

For more information about how to define a button in NetDraw, see “Defining Custom Toolbars and Buttons” on page 168.

Custom Object Events

A custom object can handle several events. The event and the function to call when the event occurs are defined in the configuration file, and the actual function is defined in a script file included in NetDraw, such as script/customobjects.js. Several default NetDraw objects have event handlers defined in script/customobjects.js, which serve as examples.

The following table lists the events that are available for handling. The oObj and oCustomObject handler parameters are present for all events (except oninitialize, for which only oObj is present), and refer to the source object and the JScript object (refer to script/configobjects.js for details) representing the custom object configuration, respectively.

<table>
<thead>
<tr>
<th>Event Name</th>
<th>Handler Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>onplace</td>
<td>• oObj&lt;br&gt;• oCustomObject&lt;br&gt;• bInsClipBoard: true if the event is a result of a paste operation&lt;br&gt;• bInsSymbol: true if the event is a result of inserting a symbol from the symbol library</td>
<td>Called after an object has been inserted, positioned, and then placed.</td>
</tr>
<tr>
<td>onanchortoggle</td>
<td>• oObj&lt;br&gt;• oCustomObject&lt;br&gt;• bShow: true if the anchors were toggled to be visible</td>
<td>Called when the anchors for an object are toggled (hidden/shown); typically when the object is being selected/cleared.</td>
</tr>
<tr>
<td>Event Name</td>
<td>Handler Parameters</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>--------------------------------------------------------------------------------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| onsettextcolor  | • oObj  
• oCustomObject  
• SetTextColor: the text color                                                    | Called when the text color for an object is changed.                          |
| ondblclick      | • oObj  
• oCustomObject  
• oEvent: a reference to the IE event object                                      | Called when an object is double-clicked.                                     |
| onsetcursor     | • oObj  
• oCustomObject  
• Style: the value of style.cursor                                                     | Called when the mouse cursor is placed on an object.                         |
| oninitialize    | • oObj                                                                 | Called when the object is loaded in run mode. The requiresObjectHTC attribute may need to be set to true in the object configuration. |
| onupdateid      | • oObj  
• oCustomObject  
• oAnchorSet: the object’s anchor set (a span containing up to 8 divs, corresponding to the object’s anchors)  
• sNewID: the new object ID  
• sOldID: the old ID                                                           | Called when an object’s ID is updated.                                        |
Extending the Context Menu for a Custom Object

In design mode, you can extend the context menu of a selected object. Menu items can be added above the default items by defining an `<ExtendContextMenu>` element and children `<MenuItem>` elements in the configuration file as defined by the DTD.

The `onMenuClick` attribute on the `ExtendContextMenu` element defines the function called when the user clicks on one of the extended menu items. This function receives the following parameters:

- `oObj`: See [Custom Object Events].
- `oCustomObject`: See [Custom Object Events].
- `sMenuItem`: A string representation of the text for the menu item that was clicked.
- `sPath`: A string representation of the path to the menu item that was clicked (not used in this version, since menu items cannot be nested).

Non-Visual Objects

Non-visual objects are interfaces that can be used by other objects. For example, you can implement a database connection as a non-visual component so that other objects can use the connection during run mode, although the icon for the non-visual database connection would only be visible in design mode.

Non-visual objects are configured through the Property Inspector, and may or may not require property editors for their configuration, depending on the complexity of the non-visual object’s configuration requirements.

In design mode, the non-visual object is represented as an icon that can be selected and moved, but not resized. In run mode, the icon is hidden, and an HTC representing the non-visual object’s functionality is added to the object. This HTC has an onload event, and exposes an interface that other objects can use.

**FMI**

For more information, see “Working with Property Editors” on page 174.

Writing the HTC for a Non-Visual Object

All HTCs written for run mode interfaces should have an `ondocumentready` event, which ensures that NetDraw is in run mode. This can be detected by the following code:

```javascript
if (!document.body.designMode) {
    // we are in run mode
} //if
```

Events can be defined and generated within the HTC. Events defined for a behavior do not bubble, they execute on the element to which the behavior is attached. You can use this mechanism to set up event notification. For example, you can configure a database query object in design mode to be associated with a database connection object. The query object could then attach to an `onconnect` event.

```javascript
onconnect: function(sPath) {
    // your code here
}
```
event exposed by the connection object in run mode, and the query object could then run its query and generate its own ondata event. Any code that attaches to events on other objects should be executed in the ondocumentready event, not the oncontentready event.

For examples, see the HTCs in the script/components directory.

---

**FMI**

---

**Advanced Custom Objects**

This section contains additional information that explains how to make more complicated custom objects. These features are used by at least one default custom object already. For an additional reference, see script/customobjects.js.

**Changing Object Behavior According to Mode**

An object can determine if it is in run mode or design mode. This is useful if the object needs to act or look differently in each mode. For example, a NetDraw button object can be disabled in design mode, but enabled in run mode. The following code example shows how to differentiate between run mode and design mode.

```javascript
if (document.body.designMode) {
    // we are in design mode
} else {
    // we are in run mode
} //if
```

**Forcing an Object to Always have object.htc Applied in Run Mode**

The object.htc HTC is removed from all static objects to decrease load time in run mode. There are a few instances where object.htc will not be removed:

- When the object has stored data, such as when the object is animated.
- When the object has requiresObjectHTC set to T in the object configuration.

The requiresObjectHTC attribute is available so that objects can have initialization code executed when they are loaded in run mode. If the object does not store data and it has an oninitialize event defined for it, then it must have the requiresObjectHTC attribute set to T in order for the event to occur.
Preventing Grouping Custom Objects

Only objects contained in a VML wrapper can be grouped. This is not a problem in design mode, where all objects must be VML or in a VML wrapper. However, it is possible that the VML object in design mode could be used as a placeholder for an applet or some other object that does not work properly inside a VML wrapper.

The trend is an example in which the design mode object gets replaced by a different object in run mode. Because an applet does not work properly inside of a VML wrapper, you must prevent the trend object from being grouped with other objects in design mode. This prevents unexpected object placement in run mode: if a trend was allowed to be grouped with other objects and then the group was moved, resized, or rotated, the actual placement of the applet in run mode would be different than the placement of the placeholder in design mode.

To prevent an object from being grouped with other objects, set the ungroupable attribute to T in the object configuration.

Using the AutoSize Feature

The AutoSize feature attempts to size the object boundaries (and anchor sets) to the actual size of a child object specified in the object configuration. This means that instead of a preset width and height for all objects of a certain type, the width and height can be dynamic, based on object content.

For example, the text object can hold text of varying width and height, depending on how much text the user types. By adding the AutoSize element to the text object's configuration, the boundaries of the text object are calculated to fit the actual volume of the text.

The AutoSize feature is only used when a custom object is first created. It will not recalculate the object boundaries if the object's contents are changed after object creation in design mode.

The calculated width and height for an object may be off by a few pixels owing to an unknown margin or padding. To compensate for these offsets, you can increase the calculated width and height by specifying values for the PaddingWidth or PaddingHeight attributes on the AutoSize element in the object configuration.

The AutoSize feature calculates the object boundaries by using the actual rendered dimensions of a child object within the custom object. By default, it uses the dimensions of the first child of the custom object (it ignores VML subelements such as stroke and fill). However, you can extend this to specify that AutoSize should use another child element.

For example, the text object specifies .children(0) for the AutoSizeTargetPath attribute of the AutoSize element in the object configuration. This causes the text object to be resized to the actual dimensions of the first child of the child of the text object, for example, getFirstHTMLChild(Text1).children(0).

Preventing Object Resizing

You can prevent users from resizing custom objects in design mode by specifying LockHeight= ‘true’ or LockWidth= ‘true’ in the actual HTML definition.
of the custom object. These must be placed as expandos on the main VML/VML wrapper object.

You can lock both width and height, or only one of the dimensions. For an example on how to specify LockHeight and LockWidth, see the getNonVisualHTML() function in the script/customobjects.js file.

Getting Information about NetDraw Using Functions

When coding script to handle object events, there are several functions available to make common tasks easier. The following table lists some of these functions.

**NetDraw Functions**

<table>
<thead>
<tr>
<th>Function</th>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getObjects()</td>
<td>n/a</td>
<td>Returns a collection of all objects with object.htc applied (all objects are returned when in design mode).</td>
</tr>
<tr>
<td>doSelect()</td>
<td>oObj</td>
<td>Selects the object that was passed in.</td>
</tr>
<tr>
<td>doUnselect()</td>
<td>oObj</td>
<td>Clears the object that was passed in.</td>
</tr>
<tr>
<td>doSelectAll()</td>
<td>aObj [optional]</td>
<td>Selects all of the objects, or the array that was passed in, if present.</td>
</tr>
<tr>
<td>doUnselectAll()</td>
<td>aObj [optional]</td>
<td>Clears all of the objects, or the array that was passed in, if present.</td>
</tr>
<tr>
<td>doDelete()</td>
<td>n/a</td>
<td>Deletes all of the selected objects.</td>
</tr>
<tr>
<td>Function</td>
<td>Parameters</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>doDirty()</td>
<td>bDirty</td>
<td>Sets the document’s dirty status (saved state) to bDirty.</td>
</tr>
<tr>
<td>exists()</td>
<td>sID</td>
<td>Returns true if an object with sID exists.</td>
</tr>
<tr>
<td>getTopObj()</td>
<td>oObj</td>
<td>Returns the topmost NetDraw object, if oObj is a subelement of a group or a subelement of an object.</td>
</tr>
<tr>
<td>getUsername()</td>
<td>n/a</td>
<td>Returns the user name of the logged-on Operational Insight user.</td>
</tr>
<tr>
<td>doAlert()</td>
<td>sMessage, oWin [optional]</td>
<td>Displays a dialog box that shows the sMessage, and which is modal to oWin.</td>
</tr>
</tbody>
</table>
Chapter 15: Tutorial

Building an Application

Operational Insight is an extensible system that enables programmers to create new applications by reusing existing Operational Insight components. This tutorial shows you how to build a new application within Operational Insight that:

- Makes use of the Operational Insight content database.
- Can load and save its content.
- Uses many of the concepts discussed in this guide.

The application created in this tutorial is a simple text editor application. Users of this application can enter text into an edit box, save the text to Operational Insight, and reload the text, either from the tree or from the application itself.

Operational Insight Database Overview

You must have a basic understanding of the Operational Insight database structure to develop new applications. Specifically, you must understand three components of the database:

The
- Content database, in which Operational Insight stores all application data.
- Tools and Operational Insight applications.
- User permissions, which determine who has access to the application content.

Content Database

The content database stores all application-generated data, such as trend configurations and NetDraw pages. The content database can also contain other files, such as simple HTML or Excel files.

Using NetManage, users and administrators can view, edit, and delete content from the content database.

FMI

For more information about administering the content database, see “Managing the Operational Insight Database” in the Operational Insight Administrator's Guide.

Tools

Operational Insight uses tools to define and separate content from different applications. NetTrend (trend builder) and NetDraw are examples of tools that are included with Operational Insight.
Each tool has an associated template file. When a user requests content for an application, Operational Insight merges the application data with the template file, and then sends the end result to the Web browser.

FMI
For more information about tools, see “Using Common Operational Insight Tools” in the Operational Insight User’s Guide.

User Permissions
When an application saves content to the database, only specific users have access to the data. Permissions to view the saved content can be set when the file is saved, or after it is in the database, using NetManage.

FMI
For more information about administering user permissions, see “Setting User and Group Permissions” in the Operational Insight Administrator’s Guide.

Creating the Application Tool
Before you can load and save content to a Operational Insight database, you must create a tool in Operational Insight with which the content will be associated. The following procedure creates a basic tool that you can configure later to properly handle the application’s content.

To create a tool in Operational Insight

1. Start NetManage.
2. Log on to NetManage as an administrative user.
3. On the left pane of NetManage, click Tools.
4. Click Add to launch the tool wizard.
5. Enter the tool description as My Notepad.
6. Enter the tool name as NPAD.
7. Click Next, and then select the Allow Shortcuts check box.
8. Click Finish, and when a warning about an invalid path appears, click Yes.

FMI
For more information about using NetManage, see “Using the NetManage Administrator Console” in the Operational Insight Administrator’s Guide.
Creating the Application Editor

For this simple application, create an HTML page that contains a text edit area and a menu for loading and saving text.

Example: A basic text editor page (TextEditor.html).

```html
<HTML>
<HEAD>
<TITLE>My Notepad</TITLE>
</HEAD>
<BODY>
<H2>My NotePad</H2>
<TextAREA id=taText rows=25 cols=80>
</TEXTAREA>
</BODY>
</HTML>
```

Loading and Saving

You must create scripts and include standard Operational Insight scripts in order to load and save the text for the editor. The dialog boxes required are contained in PNDialogs.js. You must also include a WebDataSet applet on the page to allow loading and saving.

Example: Adding load and save functionality (LoadSave.html).

```html
<HTML>
<HEAD>
<TITLE>My Notepad</TITLE>
<SCRIPT src=/ProcessNet/html/dialogs/scr/pndialogs.js>
</SCRIPT>
<SCRIPT defer=true>
function btnLoad.onclick() {
    var sText = OpenLoadFileDialog("NPAD", "LoadText", "Enter a file to load", WebDS);
    if(sText != null)
        taText.value = sText[6];
}

function btnSave.onclick() {
    var aRet = OpenSaveFileDialog("NPAD", taText.value, "Save Text", "Enter a filename", WebDS);
    if(aRet[0] != 1)
        ErrorMessage(aRet[1]);
}
</SCRIPT>
</HEAD>

<BODY>
<H2>My NotePad</H2>
<BUTTON id=btnLoad>Load</BUTTON>
<BUTTON id=btnSave>Save</BUTTON>
<br>
<TEXTAREA id=taText rows=25 cols=80>
</TEXTAREA>
<APPLET id=WebDS
    code="JWebDataSet.JWebDataSet.class"
    archive = "/ProcessNet/HTML/PNJtrend/PNJtrend.jar"
    style="Width: 0px; Height: 0px; Position: Absolute;"
Creating the Content Viewer

You can use the actual application page to load generated text, but the text is not displayed properly when you add links in the tree to the “files” saved by the application. To view the text properly, you must create a viewer page for the application.

The viewer page is a template page that NetPortal merges with the actual application data before the page is displayed. Tool template pages use replaceable parameters to define where the content is placed. The supported parameters are shown in the following table.

Parameters for Viewer Pages

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;#CONFIG_DATA&gt;</code></td>
<td>The actual application data.</td>
</tr>
<tr>
<td><code>&lt;#CONFIG_ID&gt;</code></td>
<td>The ConfigID of the content. This can be used within the template page to load the content after the page has loaded. It is useful for binary configuration data.</td>
</tr>
<tr>
<td><code>&lt;#CONFIG_NAME&gt;</code></td>
<td>The name of the file.</td>
</tr>
<tr>
<td><code>&lt;#CONFIG_DESC&gt;</code></td>
<td>A description of the file.</td>
</tr>
<tr>
<td><code>&lt;#POST_DATA&gt;</code></td>
<td>The data posted to the server, in the format param=value&amp;param2=value2</td>
</tr>
</tbody>
</table>

Building the Template Page

The template page for this application looks similar to the actual application, but does not have the ability to edit, load, or save the displayed text.

Example: The application template page (Template.html).

```html
<HTML>
  <HEAD>
    <TITLE>My Notepad (Viewer)</TITLE>
  </HEAD>
  <BODY>
    <H2>My NotePad Viewer</H2>
    <H3>Viewing `<#CONFIG_NAME>`</H3>
    <TEXTAREA id=taText rows=25 cols=80 READONLY>
      `<#CONFIG_DATA>`
    </TEXTAREA>
  </BODY>
</HTML>
```
Reconfiguring the Tool
After the template file has been created, use NetManage to associate the tool with its template file.

To associate a tool with its template file
1. Start NetManage.
2. Log on as an administrative user.
3. On the left pane in NetManage, click **Tools**.
4. Double-click **My Notepad**.
5. Click **Next** to view the edit box for the template file.
6. Browse for the template file, or enter the full path and filename in the edit box.
7. Click **Finish**.

**FMI**
For more information about using NetManage, see “Using the NetManage Administrator Console” in the *Operational Insight Administrator’s Guide*.

For these settings to take effect immediately, you must restart the World Wide Web Publishing service. If you do not restart the service, the settings will take effect automatically within one minute.

After modifying the tool configuration, any links in the tree that reference content created with the “My Notepad” application load the template page with the application text.
Chapter 16: Training Exercises

Use the exercises in this chapter to test your knowledge of the content covered in this guide.

Exercise 1: Basic HTML Page

This exercise shows you how to create a basic HTML page to practice simple layout elements. Before completing this exercise, read the chapter “Basic HTML and Scripting” on page 7.

Requirements

The page you create should look like the following figure.

![Basic HTML page example](image)

Procedure

Use a text editor such as Notepad (not a WYSIWYG editor) to create the Web page. Select All Files in the Save as type box, and save the file with an .htm or .html extension, such as Exercise1.html. The elements and attributes required to generate the page are:

- `<BODY bgcolor=...>`
- `<H1>`
- `<HR>`
- `<BR>`
- `<B>`
- `<FONT>`
**Exercise 2: Table Layout**

This exercise shows you how to create an HTML page that uses a table to lay out text. Tables are often used to justify text. Before completing this exercise, make sure you have read the chapter “Basic HTML and Scripting” on page 7.

**Requirements**

The page you create should look like the following figure.

![Table layout example](image)

**Procedure**

Use a text editor such as Notepad (not a WYSIWYG editor) to create the Web page. Select All Files in the Save as type box, and save the file with an .htm or .html extension, such as Exercise2.html. The elements and attributes required to generate the page are:

- `<TABLE>`
- `<TR>`
- `<TH>`
- `<TD COLSPAN=... ALIGN=...>`
- `<FONT>`

**Exercise 3: Using DHTML Styles**

DHTML (Dynamic HTML) styles can significantly enhance the presentation of HTML pages. This exercise provides you with practice using DHTML styles. Before completing this exercise, make sure you have read the chapter “Basic HTML and Scripting” on page 7.
Requirements

The resulting page should look like the following figure.

Exercise 4: Using PNSP for Current Values

PNSP is a simple way to retrieve data from Operational Insight. This exercise shows you how to use PNSP to retrieve snapshot data from the OPC simulation server. Before completing this exercise, make sure you have read the chapters “Basic HTML and Scripting” on page 7, and “Advanced PNSP” on page 71.

Requirements

Use PNSP (Operational Insight Server Pages) to retrieve the current values and time stamps for two tags, and display these in a table.
Chapter 16: Training Exercises

The resulting page should look like the following figure.

![Using PNSP for current values example](image)

**Procedure**

Use a text editor such as Notepad to manually create the table and PNSP elements. Use only the PNSP [#Value] element.

You should save the *.pnsp files you create in the following folder:
C:/Program Files/Matrikon/ProcessSuite/ProcessNet/web/html/Samples.

PNSP files must be accessed through the Web server and a URL, rather than directly from the hard disk drive. To view the files through the Web server, type the following URL into the Web browser’s Address edit box, replacing `{filename}` with the name of the file you want to view:
http://localhost/ProcessNet/html/Samples/{filename}

Experiment with the various attributes and display formats available in the value element.

**Exercise 5: Using PNSP for Tag History**

This exercise shows you how to use PNSP to retrieve historical values from the OPC simulation server. Before completing this exercise, make sure you have read the chapters “Basic HTML and Scripting” on page 7, and “Advanced PNSP” on page 71.

**Requirements**

Use PNSP to retrieve the historical values and timestamps for a single tag, and display these in a table generated from a [#Band] element. Display the last ten minutes of data at one-minute intervals.
Exercise 6: Using PNSP for SQL

The resulting page should look like the following figure.

![Tag: Random.Int2](image)

**Procedure**

Use a text editor such as Notepad to manually create the table and PNSP elements. Use the PNSP [#DataSet], [#Band], and [#Value] elements. You should save the *.pnsp files you create in the following folder: C:/Program Files/Matrikon/ProcessSuite/ProcessNet/web/html/Samples.

PNSP files must be accessed through the Web server and a URL, rather than directly from the hard disk drive. To view the files through the Web server, type the following URL into the Web browser’s Address edit box, replacing `{filename}` with the name of the file you want to view:

http://localhost/ProcessNet/html/Samples/{filename}

Experiment with variations of the band definition and the data set parameters.

**Exercise 6: Using PNSP for SQL**

This exercise shows you how to use PNSP to retrieve textual data from the sample MS Access database installed with Operational Insight. Before completing this exercise, make sure you have read the chapters “Basic HTML and Scripting” on page 7, and “Advanced PNSP” on page 71.

**Requirements**

Use PNSP to build a table listing all the industries defined in the database.
Chapter 16: Training Exercises

The resulting page should look like the following figure.

Using PNSP for SQL example

**Procedure**

Use a text editor such as Notepad to manually create the table and PNSP elements. The SQL query you should use for the [#DataSet] element is:

```sql
select long_name from industry
```

You should save the *.pnsp files you create in the following folder: C:/Program Files/Matrikon/ProcessSuite/ProcessNet/web/html/Samples.

PNSP files must be accessed through the Web server and a URL, rather than directly from the hard disk drive. To view the files through the Web server, type the following URL into the Web browser’s Address edit box, replacing `{filename}` with the name of the file you want to view:

http://localhost/ProcessNet/html/Samples/{filename}

Experiment with different queries and band definitions.

**Exercise 7: HTML and Scripting**

This exercise lets you practice basic scripting in HTML pages. Before completing this exercise, make sure you have read the chapter “Basic HTML and Scripting” on page 7.

**Requirements**

Create an HTML page with an entry field, a button, and a `<DIV>` element. When the button is clicked, the text within the `<DIV>` element (innerText property) is set to the text entered into the entry field.
Lay out the page as you want, optionally using a table for the entry field and button. The <DIV> element can be placed after the button, or positioned absolutely for effect.

**Procedure**

Use the <BUTTON> element to create the button, and create an event handler for it, using the button's ID. Make sure you set the <SCRIPT> element’s defer attribute to true.

### Exercise 8: URL Queries for Tag Data

Operational Insight provides users with the ability to get data by entering queries into the address box of the browser. This example lets you experiment with retrieving tag values from a URL. Before completing this exercise, make sure you have read the chapter “NetPortal Data Access Interface” on page 27.

#### Requirements

In the address edit field, enter the URL queries (only one query can be entered at a time) that will return the following data:
- The current value for Random.Int2 from OPC_SIM in HTML format.
- Ten minutes of one-minute data for Random.Int2 in CSV and Excel formats.
- The current values for Saw-toothed Waves.Int2 and Random.Int2, including the “snapshot” time stamp (use SnapTime = true in the query).

#### Procedure

Log on to Operational Insight, and then enter the URL to retrieve the data. For example:

```
http://localhost/ProcessNet/isapi/netportal/netportal.dll/data/opc_sim?tag0 = xxx
```

Be sure to specify the data format, such as format = html.

### Exercise 9: URL Queries for SQL Data

URL queries also provide a way of retrieving data using SQL. In this exercise, run a SQL query to retrieve data. Before completing this exercise, make sure you have read the chapter “NetPortal Data Access Interface” on page 27.

#### Requirements

Using the query from the PNSP SQL exercise, retrieve a list of industries from the SQL_DS data source.
Chapter 16: Training Exercises

Procedure
Log on to Operational Insight, and then enter the URL to retrieve the data. Experiment with different display formats, such as html, Excel, CSV, and ADO.

Exercise 10: HTML Forms for Data Access
URL queries are powerful, but not user friendly, and they cannot be “canned” for an application. HTML forms allow users to make data requests without being aware of the URL query syntax required. Before completing this exercise, make sure you have read the chapter “HTML Forms” on page 67.

Requirements
Create a data entry form, as shown in the following figure. When the Submit button is clicked, the data for the entered tag and time range should be displayed in an HTML table.

![HTML forms for data access example](image)

Procedure
Use a table to align the entry fields of the form. No scripting is required for this exercise.
Experiment with changing the target attribute of the <FORM> element. Add another entry field to specify the format for the data.

Exercise 11: HTML Forms for Data Entry
Forms can be used for both data access and data entry. However, data entry forms normally require additional scripting. This exercise gives you practice in creating forms and writing scripts. Before completing this exercise, make sure you have read the chapter “HTML Forms” on page 67.
Requirements

Create a form that enables the user to add entries to the industry table in the SQL_DS data source. The user must be able to enter an industry code, a short name for the industry, and a long name. (These are the fields in the industry table.) When the user clicks submit, a new record should be added to the industry table with the data specified in the form.

Procedure

This data entry form requires a script to handle the onsubmit event of the <FORM> element. Within this event, you must build the SQL statement to insert the record into the database, and assign this SQL to a hidden form field called exec. The SQL required to insert the record takes the form:

```
insert into industry values([code], [short name], [long name])
```

where [code], [short name], and [long name] are values from the form.

Remember to provide a hidden field called format so that the returned data is displayed in the Web browser.

Exercise 12: Using PNSP in Forms

PNSP can be useful for creating selection lists, such as combo boxes and list boxes. This exercise shows you how to use PNSP to generate lists of data from a database. Before completing this exercise, make sure you have read the chapters “HTML Forms” on page 67, and “Advanced PNSP” on page 71.

Requirements

Create a form that enables the user to select a company name from a drop-down list box (combo box). When the Submit button is clicked, a list of all orders for the selected customer should be displayed in an HTML table.

Procedure

The drop-down list for the form should be generated from PNSP [#DataSet] and [#Band] elements. The page requires a script to handle the onsubmit event of the <FORM> element. Within this event, build the SQL statement that retrieves the data from the orders table, and assign this SQL to a hidden form field called sql. The SQL statements required for this exercise are:

```
Combo box: select CustNo, Company from Customer
Table: select * from orders where CustNo = [CustNo]
```

where [CustNo] is the customer number of the selected company name. The first SQL statement (for the [#Band] element) retrieves the company name for display, and the customer number for use as the combo box’s value.

Remember to provide a hidden field called format, so that the returned data is displayed in the Web browser.
Exercise 13: Accessing Data Using the WebDataSet

The WebDataSet is the primary method for accessing data programmatically. This exercise shows you how to use the WebDataSet to retrieve and sum tag data. Before completing this exercise, make sure you have read “Using the WebDataSet” on page 95.

Requirements

Use the `openRecordset` method of the WebDataSet to retrieve the data, and check for errors returned from the server. Once the data has been retrieved, iterate through the values and calculate the total, and then display the total using the `alert` function.

Procedure

Create a page that contains a button and the WebDataSet applet. Make sure you set the size of the applet to zero, and position it as `absolute`.

To iterate through the values returned from the `openRecordset` call, use a `while` loop, similar to:

```javascript
var fldVal = rsResult.FieldByName("Random.Int2.Value");
rsResult.First();
while(rsResult.Eof() != true)
{
    fTotal += fldVal.AsFloat();
    rsResult.Next();
} //while
alert(fTotal);
```

Additional Exercises

- Modify the query so that an error is generated (for example, enter an invalid tag name).
- Modify the script to show the average value for the ten-minute interval.
- Start a new instance of Internet Explorer and, without logging on to Operational Insight, run the page.

Exercise 14: Accessing Real-Time Data

This exercise shows you how to use the DataHub JavaScript library to access real-time data from the server. Before completing this exercise, make sure you have read the section “DataHub Object” on page 129.

Requirements

Create a Web page that displays the last value for the `Random.Int2` tag in the OPC simulation data source. The value should start updating as soon as the page is loaded, and refresh every second. Display the value beside a text label, for example:

Tag Value: ####
Exercise 15: Accessing Non-Tag Data

Procedure

Use the document’s `onload` event to create a DataHub object, and use the object to access the tag’s data at one-second intervals. When the new data arrives, set the `innerText` property of a `<div>` element on the page to display the value.

Exercise 15: Accessing Non-Tag Data

The WebDataSet can be used to retrieve data other than tag values, such as SQL data, tag lists, data source lists, and other metadata. Before completing this exercise, make sure you have read the section “Using the WebDataSet” on page 95.

Requirements

This exercise shows you how to dynamically generate an HTML table when the user clicks a button on the page. The table should contain the results of a SQL query.

Procedure

Create a page that contains a button, and in the button’s event handler, open a record set, using a WebDataSet applet with the following query:

```javascript
sql = select * from customer
```

Once the data has been retrieved, dynamically build an HTML table for it.

To do this, use a combination of a `while` loop and a `for` loop. Within these loops, generate a string that contains the table definition. Finally, insert the string into the document.

For example:

```javascript
var sTable = "<TABLE border = 1>"
rsResult.First();
while( rsResult.Eof() != true )
{
    sTable += "<tr>";
    for(var nFld = 0; nFld < sResult.FieldCount(); nFld++)
    {
        sTable += "<td>" +
            rsResult.Fields(nFld).AsString() +
        "</td>";
    } //for
    sTable += "</tr>";
    rsResult.Next();
} //while
sTable += "</table>";
document.body.insertAdjacentHTML('beforeEnd', sTable);
```

Additional Exercises

- Modify the query to retrieve a list of data sources.
- Modify the query to retrieve a list of tags.
Exercise 16: Adding Data to a Trend

This exercise shows you how to use the trend applet to display data generated by a script. Before completing this exercise, make sure you have read the section “NetTrend Programming Interface” on page 103.

Requirements
When the user clicks a button on the page for this exercise, a new data series should be added to the trend on the page. This data should be generated by the script, and not retrieved from the server.

Procedure
Create a page that contains a button and a trend applet. When the button is clicked, use the chart object's `addSeries` method to add a series to the trend. Then use the series object's `AddXYStatus` method to add a few data points to the trend.

Exercise 17: Adding Server Data to a Trend

This exercise shows you how to use the trend applet to display data obtained from the server. Before completing this exercise, make sure you have read the section “NetTrend Programming Interface” on page 103.

Requirements
The user should be able to enter a tag name into an edit box so that, when a button is clicked, data for the entered tag is displayed on the trend.

Procedure
Create a page that contains an edit box, a button, and a trend applet. When the button is clicked, use the chart object's `addSeries` method to add a series to the trend that specifies the tag name entered by the user. Set the parameters in `addSeries` to request data for the last ten minutes.

Exercise 18: Creating a Tabbed Dialog Box

Tabbed dialog boxes are useful for organizing information, especially data entry form categories. This exercise shows you how to create a tabbed dialog box. Before completing this exercise, make sure you have read the section “Other Operational Insight Objects” on page 129.

Requirements
When the user clicks a button on the page for this exercise, a new data series should be added to the trend on the page. This data should be generated by the script, and not retrieved from the server.
Exercise 19: Creating an Active Server Page

Procedure

Create a page that contains a `<DIV>` element. Apply the `tabsheet.htc` behavior to the `<DIV>` element and embed three tab page `<DIV>` elements within the main tabbed dialog box `<DIV>` element. On the first page of the tabbed dialog box, copy the text from the body of Exercise 1: Basic HTML Page. For the second and third pages of the dialog box, embed the HTML from the previous trend examples.

Copy and paste the body of the previous examples into the individual `<DIV>` elements for the tab pages. Copy the scripts used in the previous exercises into a script block on the tabbed dialog box page.

Exercise 19: Creating an Active Server Page

Active Server Pages are commonly used with IIS to generate HTML pages from the server. Using NetDAC, Operational Insight developers can access data from ASP. Before completing this exercise, make sure you have read the section “NetDAC Server-side Data Access” on page 149.

Requirements

Each time the page loads, a table containing the values and time stamps for a tag, `Random.Int2` should be generated. The concepts used in the “Accessing Non-Tag Data” exercise should be used.

Procedure

Create a Web page file named with the extension `.ASP`. Within this page, embed the code required to generate the table. The script must access the NetDAC object and log on using an existing user session. Once logged on, use NetDAC to retrieve the data for the tag, and then use the `Response.Write` method to generate the table.
Appendix A: NetObjects Stored Procedures

This appendix outlines the stored procedures that are available for NetObjects. Currently, all procedures must be accessed through Operational Insight. They are used just like any other query in Operational Insight. For example, to add a new user, the Operational Insight query would be as follows:

```sql
dp_addUser 'username', 'password', 1
```

All procedures are available to administrators, but only a small subset of the procedures is available to standard users:

- `dp_getRelationshipTypeInfo`
- `dp_getImageInfo`
- `dp_getAttributeInfo`
- `dp_getAttrStateInfo`
- `dp_getAttributeGroupInfo`
- `dp_getObjectTypeInfo`
- `dp_getObjectInfo`
- `dp_getObjectTree`
- `dp_getAllData`
- `dp_getAttributeData`
- `dp_setAttributeData`

NetObjects Stored Procedures

User Subsystem

dp_addUser

This procedure adds a user to NetObjects.

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>userName</td>
<td>The username for the given user.</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>password</td>
<td>This user's password.</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>isAdmin</td>
<td>Whether this person will be an administrator.</td>
<td>Boolean</td>
<td></td>
</tr>
</tbody>
</table>
Result Set

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>USER_ID</td>
<td>The ID of the newly created user.</td>
<td>Integer</td>
</tr>
</tbody>
</table>

**dp_removeUser**

This procedure removes a user from NetObjects.

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>userId</td>
<td>The user to remove.</td>
<td>Integer</td>
<td></td>
</tr>
</tbody>
</table>

**dp_editUserInfo**

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>userId</td>
<td>The user to edit the info for.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>userName</td>
<td>The username for the user.</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>password</td>
<td>The password for this user.</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>isAdmin</td>
<td>Whether the user will be an administrator.</td>
<td>Boolean</td>
<td></td>
</tr>
</tbody>
</table>

**dp_getUserInfo**

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>userId</td>
<td>The user to retrieve information for. Default = 0 (retrieve all users).</td>
<td>Integer</td>
<td>yes</td>
</tr>
</tbody>
</table>
Result Set

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>USER_ID</td>
<td>The ID for the user.</td>
<td>Integer</td>
</tr>
<tr>
<td>USERNAME</td>
<td>The user name.</td>
<td>String</td>
</tr>
<tr>
<td>PASSWORD</td>
<td>The user's password.</td>
<td>String</td>
</tr>
<tr>
<td>IS_ADMIN</td>
<td>Whether this user is an administrator.</td>
<td>Boolean</td>
</tr>
</tbody>
</table>

Relationship Type Subsystem

dp_addRelationshipType
This procedure adds a new relationship to NetObjects.

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>relTypeName</td>
<td>The name of the relationship type.</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>strength</td>
<td>The strength of the relationship type. (1 = Weak, 2 = Strong).</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>relTypeDescription</td>
<td>The description of the relationship type.</td>
<td>String</td>
<td></td>
</tr>
</tbody>
</table>

Result Set

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>RELATIONSHIP_TYPE_ID</td>
<td>The ID of the newly created relationship type.</td>
<td>Integer</td>
</tr>
</tbody>
</table>

dp_removeRelationshipType
This procedure removes a relationship type from NetObjects.

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>relTypeId</td>
<td>The relationship type to remove.</td>
<td>Integer</td>
<td></td>
</tr>
</tbody>
</table>
Appendix A: Training Exercises
NetObjects Stored Procedures

dp_editRelationshipType
This procedure edits the given relationship type.

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>relTypeId</td>
<td>The relationship type to edit.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>relTypeName</td>
<td>The name of the relationship type.</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>strength</td>
<td>The strength of the relationship type.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>relTypeDescription</td>
<td>The description of the relationship type.</td>
<td>String</td>
<td>yes</td>
</tr>
</tbody>
</table>

dp_getRelationshipTypeInfo
*User Accessible*
This procedure retrieves the information for a relationship type. All relationships or relationships of a given strength can also be requested.

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>relTypeId</td>
<td>The ID of the relationship type to retrieve. Default = 0 (All relationships).</td>
<td>Integer</td>
<td>yes</td>
</tr>
<tr>
<td>strength</td>
<td>The strength of the relationship type (1= Weak, 2 = Strong). Only applies when all relationships are requested.</td>
<td>Integer</td>
<td>yes</td>
</tr>
</tbody>
</table>

Result Set

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>RELATIONSHIP_TYPE_ID</td>
<td>The ID of the relationship type.</td>
<td>Integer</td>
</tr>
<tr>
<td>NAME</td>
<td>The name of the relationship type.</td>
<td>String</td>
</tr>
<tr>
<td>DESCRIPT</td>
<td>The description of the relationship type.</td>
<td>String</td>
</tr>
<tr>
<td>STRENGTH</td>
<td>The strength of the relationship type (1= Weak, 2 = Strong).</td>
<td>Integer</td>
</tr>
</tbody>
</table>
**Image Subsystem**

**dp_addImage**
This procedure adds an image to NetObjects.

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>imageName</td>
<td>The name to use for the image.</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>imageLocation</td>
<td>The location of the image. Must be relative to the web path.</td>
<td>String</td>
<td></td>
</tr>
</tbody>
</table>

**Result Set**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMAGE_ID</td>
<td>The ID of the newly created image.</td>
<td>Integer</td>
</tr>
</tbody>
</table>

**dp_removeImage**
This procedure removes an image from NetObjects.

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>imageid</td>
<td>The ID of the image to remove.</td>
<td>Integer</td>
<td></td>
</tr>
</tbody>
</table>

**dp_editImageInfo**
This procedure edits the information for an image.

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>imageid</td>
<td>The ID of the image to edit.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>imageName</td>
<td>The name of the image.</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>imageLocation</td>
<td>The location of the image including file name.</td>
<td>String</td>
<td></td>
</tr>
</tbody>
</table>
Appendix A: Training Exercises
NetObjects Stored Procedures

**dp_getImageInfo**

*User Accessible*

This procedure retrieves the information for a given image or lists all images.

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>imageld</td>
<td>The image to retrieve. Default = 0 (all images).</td>
<td>Integer</td>
<td>yes</td>
</tr>
</tbody>
</table>

**Result Set**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMAGE_ID</td>
<td>The ID of the image.</td>
<td>Integer</td>
</tr>
<tr>
<td>NAME</td>
<td>The name of the image.</td>
<td>String</td>
</tr>
<tr>
<td>LOCATION</td>
<td>The location of the image including file name.</td>
<td>String</td>
</tr>
</tbody>
</table>

**Module Subsystem**

**dp_addModule**

This procedure adds a new module to NetObjects.

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>moduleName</td>
<td>The module name.</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>isEnabled</td>
<td>Whether the module is enabled.</td>
<td>Boolean</td>
<td></td>
</tr>
<tr>
<td>moduleDescription</td>
<td>The module description.</td>
<td>String</td>
<td>yes</td>
</tr>
<tr>
<td>componentName</td>
<td>The name of the back end component in Windows.</td>
<td>String</td>
<td>yes</td>
</tr>
</tbody>
</table>

**Result Set**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>MODULE_ID</td>
<td>The ID of the newly created module.</td>
<td>Integer</td>
</tr>
</tbody>
</table>

---

Page 212  Operational Insight R362.1 Programmer’s Guide
dp_removeModule
This procedure removes a module from NetObjects.

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>moduleId</td>
<td>The ID of the module to remove.</td>
<td>integer</td>
<td></td>
</tr>
</tbody>
</table>

dp_getModuleInfo
This procedure will retrieve the information for a given module or list all modules.

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>moduleId</td>
<td>The module to retrieve. Default = 0 (get all modules).</td>
<td>Integer</td>
<td>yes</td>
</tr>
</tbody>
</table>

dp_editModuleInfo
This procedure changes the information for a given module.

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>moduleId</td>
<td>The ID of the module to edit.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>moduleName</td>
<td>The name of the module.</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>enabled</td>
<td>Whether the module is enabled.</td>
<td>Boolean</td>
<td></td>
</tr>
<tr>
<td>moduleDescription</td>
<td>The module description.</td>
<td>String</td>
<td>yes</td>
</tr>
<tr>
<td>componentName</td>
<td>The module backend component.</td>
<td>String</td>
<td>yes</td>
</tr>
</tbody>
</table>
**dp_addModuleView**

This procedure adds a view to the given module.

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>moduleId</td>
<td>The module to add the view to.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>viewName</td>
<td>The name of the new view.</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>page</td>
<td>The location of the web page.</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>imageIndex</td>
<td>The number of the image to use.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>viewDescription</td>
<td>The description of the view.</td>
<td>String</td>
<td>yes</td>
</tr>
</tbody>
</table>

**Result Set**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>VIEW_ID</td>
<td>The ID of the newly created view.</td>
<td>integer</td>
</tr>
</tbody>
</table>

**dp_removeModuleView**

This procedure will remove a given view for a module.

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>moduleId</td>
<td>The module that the view belongs to.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>viewId</td>
<td>The view to remove.</td>
<td>Integer</td>
<td></td>
</tr>
</tbody>
</table>

**dp_editModuleView**

This procedure will change the information for a given view.

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>moduleId</td>
<td>The module that the view belongs to.</td>
<td>Integer</td>
<td></td>
</tr>
</tbody>
</table>
dp_getModuleViewInfo
This procedure retrieves the info for a given view or lists the views.

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>moduleId</td>
<td>The module that the view belongs to. Default = 0 (get all modules).</td>
<td>Integer</td>
<td>yes</td>
</tr>
<tr>
<td>viewId</td>
<td>The view to get information for Default = 0 (get all views for this module).</td>
<td>Integer</td>
<td>yes</td>
</tr>
</tbody>
</table>

Result Set

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>VIEW_ID</td>
<td>The ID of the view.</td>
<td>Integer</td>
</tr>
<tr>
<td>NAME</td>
<td>The view name.</td>
<td>String</td>
</tr>
<tr>
<td>DESCRIPT</td>
<td>The view description.</td>
<td>String</td>
</tr>
<tr>
<td>PAGE</td>
<td>The page name.</td>
<td>String</td>
</tr>
<tr>
<td>IMAGE_ID</td>
<td>The index of this view's image.</td>
<td>Integer</td>
</tr>
</tbody>
</table>
Appendix A: Training Exercises

NetObjects Stored Procedures

Page 216

Operational Insight R362.1 Programmer’s Guide

dp_addViewCollectionAssoc
This procedure adds a view collection association.

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>moduleId</td>
<td>The module that the view belongs to. Default = 0 (get all modules).</td>
<td>Integer</td>
<td>yes</td>
</tr>
<tr>
<td>viewId</td>
<td>The view to get information for Default = 0 (get all views for this module).</td>
<td>Integer</td>
<td>yes</td>
</tr>
<tr>
<td>collectionId</td>
<td>The collection to get information for.</td>
<td>Integer</td>
<td></td>
</tr>
</tbody>
</table>

dp_removeViewCollectionAssoc
This procedure removes a view collection association.

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>moduleId</td>
<td>The module that the view belongs to. Default = 0 (get all modules).</td>
<td>Integer</td>
<td>yes</td>
</tr>
<tr>
<td>viewId</td>
<td>The view to get information for Default = 0 (get all views for this module).</td>
<td>Integer</td>
<td>yes</td>
</tr>
<tr>
<td>collectionId</td>
<td>The collection to get information for.</td>
<td>Integer</td>
<td></td>
</tr>
</tbody>
</table>

dp_getViewCollectionAssoc
This procedure retrieves the info for a given view – collection association.

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>moduleId</td>
<td>The module that the view belongs to. Default = 0 (get all modules).</td>
<td>Integer</td>
<td>yes</td>
</tr>
<tr>
<td>viewId</td>
<td>The view to get information for Default = 0 (get all views for this module).</td>
<td>Integer</td>
<td>yes</td>
</tr>
</tbody>
</table>
Result Set

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>COLLECTION_ID</td>
<td>The ID of the collection.</td>
<td>Integer</td>
</tr>
<tr>
<td>NAME</td>
<td>The collection name.</td>
<td>String</td>
</tr>
<tr>
<td>DESCR</td>
<td>The collection description.</td>
<td>String</td>
</tr>
</tbody>
</table>

**dp_addModuleMethod**
This procedure will add a method to a given view.

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>moduleId</td>
<td>The ID of the module to add to.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>methodName</td>
<td>The name of the new method.</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>methodDescription</td>
<td>The description of the method.</td>
<td>String</td>
<td>yes</td>
</tr>
</tbody>
</table>

Result Set

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>METHOD_ID</td>
<td>The ID of the method that was added.</td>
<td>Integer</td>
</tr>
</tbody>
</table>

**dp_removeModuleMethod**
This procedure removes a given method.

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>moduleId</td>
<td>The module to remove from.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>methodId</td>
<td>The method to remove.</td>
<td>Integer</td>
<td></td>
</tr>
</tbody>
</table>
dp_editModuleMethod
This procedure edits the information for a given method.

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>moduleId</td>
<td>The ID of the module that the method belongs to.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>methodId</td>
<td>The ID of the method.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>methodName</td>
<td>The name of the method.</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>methodDescription</td>
<td>The description of the method.</td>
<td>String</td>
<td>yes</td>
</tr>
</tbody>
</table>

dp_getModuleMethodInfo
This procedure will retrieve the information for a given method or lists the available methods.

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>moduleId</td>
<td>The module that the method belongs to. Default = 0 (for all modules).</td>
<td>Integer</td>
<td>yes</td>
</tr>
<tr>
<td>methodId</td>
<td>The method to retrieve. Default = 0 (for all methods that belong to the given module).</td>
<td>Integer</td>
<td>yes</td>
</tr>
</tbody>
</table>

Result Set

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>METHOD_ID</td>
<td>The ID of the method.</td>
<td>Integer</td>
</tr>
<tr>
<td>NAME</td>
<td>The name of the method.</td>
<td>String</td>
</tr>
<tr>
<td>DESCR</td>
<td>The description of the method.</td>
<td>String</td>
</tr>
</tbody>
</table>
dp_addModuleCollection
This procedure will add a module/collection relationship.

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>moduleId</td>
<td>The module that the method belongs to Default = 0 (for all modules).</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>collection name</td>
<td>The collection name you want to add the relationship to.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>collection description</td>
<td>The collection's description.</td>
<td>String</td>
<td>yes</td>
</tr>
</tbody>
</table>

Result Set

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>COLLECTION_ID</td>
<td>The ID of the collection.</td>
<td>Integer</td>
</tr>
</tbody>
</table>

dp_removeModuleCollection
This procedure will remove a module/collection relationship.

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>moduleId</td>
<td>The module that the method belongs to Default = 0 (for all modules).</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>collectionId</td>
<td>The collection id you want to add the relationship to.</td>
<td>Integer</td>
<td></td>
</tr>
</tbody>
</table>
**dp_editModuleCollection**
This procedure will edit a module/collection relationship.

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>moduleId</td>
<td>The module that the method belongs to Default = 0 (for all modules).</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>collectionId</td>
<td>The collection id you want to add the relationship to.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>collection name</td>
<td>The collection’s name.</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>collection description</td>
<td>The collection's description.</td>
<td>String</td>
<td>yes</td>
</tr>
</tbody>
</table>

**dp_getModuleCollectionInfo**
This procedure will retrieve the information for a given Module Collection.

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>moduleId</td>
<td>The module that the method belongs to Default = 0 (for all modules).</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>collectionId</td>
<td>The collection id you want to add the relationship to.</td>
<td>Integer</td>
<td></td>
</tr>
</tbody>
</table>

**Result Set**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>MODULE_ID</td>
<td>The ID of the module.</td>
<td>Integer</td>
</tr>
<tr>
<td>COLLECTION_ID</td>
<td>The ID of the collection.</td>
<td>Integer</td>
</tr>
<tr>
<td>NAME</td>
<td>The name of the collection.</td>
<td>String</td>
</tr>
<tr>
<td>DESCR</td>
<td>The description of the collection.</td>
<td>String</td>
</tr>
</tbody>
</table>
dp_getCollObjectTypes
This procedure will retrieve the object type information for a given collection.

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>collectionId</td>
<td>The collection id you want to add the relationship to.</td>
<td>Integer</td>
<td></td>
</tr>
</tbody>
</table>

Result Set

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>OBJECT_TYPE_ID</td>
<td>The ID of the object type.</td>
<td>Integer</td>
</tr>
<tr>
<td>NAME</td>
<td>The name of the collection.</td>
<td>String</td>
</tr>
<tr>
<td>DESC</td>
<td>The description of the collection.</td>
<td>String</td>
</tr>
<tr>
<td>INHERITED_FROM_ID</td>
<td>The ID of the object type that this one is inherited from.</td>
<td>Integer</td>
</tr>
<tr>
<td>TREE_ENABLED</td>
<td>Whether this object type is tree enabled.</td>
<td>Boolean</td>
</tr>
<tr>
<td>IMAGE_ID</td>
<td>The Image Id of the object.</td>
<td>Integer</td>
</tr>
</tbody>
</table>

dp_getCollAttrGroups
This procedure will retrieve the attribute group information for a given Collection.

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>collectionId</td>
<td>The collection ID you want to add the relationship to.</td>
<td>Integer</td>
<td></td>
</tr>
</tbody>
</table>

Result Set

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTRIBUTE_GROUP_ID</td>
<td>The ID of the object type.</td>
<td>Integer</td>
</tr>
<tr>
<td>NAME</td>
<td>The name of the collection.</td>
<td>String</td>
</tr>
<tr>
<td>DESC</td>
<td>The description of the collection.</td>
<td>String</td>
</tr>
</tbody>
</table>
Appendix A: Training Exercises
NetObjects Stored Procedures

dp_addMethodCollectionAssoc
This procedure associates an attribute group to a given view.

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>moduleId</td>
<td>The module that the view belongs to.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>viewId</td>
<td>The view to associate to.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>collectionId</td>
<td>The collection to associate to the view.</td>
<td>Integer</td>
<td></td>
</tr>
</tbody>
</table>

dp_removeMethodCollectionAssoc
This procedure associates an attribute group to a given view.

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>moduleId</td>
<td>The module that the view belongs to.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>viewId</td>
<td>The view to associate to.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>collectionId</td>
<td>The collection to associate to the view.</td>
<td>Integer</td>
<td></td>
</tr>
</tbody>
</table>

dp_getMethodCollectionAssoc
This procedure lists the attribute group associations for a given module.

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>moduleId</td>
<td>The module that the view belongs to.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>methodId</td>
<td>The method to retrieve the associations for.</td>
<td>Integer</td>
<td></td>
</tr>
</tbody>
</table>
Result Set

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>COLLECTION_ID</td>
<td>The collection ID.</td>
<td>Integer</td>
</tr>
<tr>
<td>NAME</td>
<td>The name of the attribute group.</td>
<td>String</td>
</tr>
<tr>
<td>DESCR</td>
<td>The description of the attribute group.</td>
<td>String</td>
</tr>
</tbody>
</table>

dp_addModuleObjectTypeAssoc
This procedure associates an ObjectType with a Module. The module will receive the events specified in the INetObjectAuditing interface.

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectTypeId</td>
<td>The object type to create the association for.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>methodId</td>
<td>The module to create the association for.</td>
<td>Integer</td>
<td></td>
</tr>
</tbody>
</table>

dp_removeModuleObjectTypeAssoc
This procedure dissociates an ObjectType and a Module. The module will no longer receive the events specified in the INetObjectAuditing interface.

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectTypeId</td>
<td>The object type to remove the association for.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>MethodId</td>
<td>The module to remove the association for.</td>
<td>Integer</td>
<td></td>
</tr>
</tbody>
</table>

dp_getModuleObjectTypeAssocInfo
This procedure returns the association between an ObjectType and all Modules.

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectTypeId</td>
<td>The object type to remove the association for.</td>
<td>Integer</td>
<td></td>
</tr>
</tbody>
</table>
Appendix A: Training Exercises

NetObjects Stored Procedures

Result Set

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>MODULE_ID</td>
<td>The module ID.</td>
<td>Integer</td>
</tr>
<tr>
<td>LEVELS_UP</td>
<td>The number of levels up the tree to start.</td>
<td>Integer</td>
</tr>
<tr>
<td>OBJECT_TYPE_ID</td>
<td>The object type to start at if found at the desired level.</td>
<td>Integer</td>
</tr>
</tbody>
</table>

Attribute Subsystem

dp_addAttribute
This procedure adds a new attribute.

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>attrName</td>
<td>The name of the new attribute.</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>dataType</td>
<td>The data type of the attribute. Can be one of: BLOB='B', EXTERNAL='E', VALUE='V', STATE='S', NUMBER='N', TABLE='T', DATASOURCE='D', DATA_COLUMN='C', KEY_COLUMN='K'.</td>
<td>Character</td>
<td></td>
</tr>
<tr>
<td>attrDescr</td>
<td>The description for this attribute.</td>
<td>String</td>
<td>yes</td>
</tr>
<tr>
<td>defaultValue</td>
<td>The default value to use when creating new instances of the attribute. Not applicable for attributes of type state until states have been created. Therefore, does not apply in the add function.</td>
<td>String</td>
<td>yes</td>
</tr>
</tbody>
</table>

Result Set

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTR_ID</td>
<td>The ID of the newly created attribute.</td>
<td>Integer</td>
</tr>
</tbody>
</table>
dp_removeAttribute
This procedure removes an attribute. This will also remove all instances where this attribute exists including the associations to groups as well as the instances that belong to objects.

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>attrId</td>
<td>The ID of the attribute to remove.</td>
<td>Integer</td>
<td></td>
</tr>
</tbody>
</table>

dp_setDefaultValue
This procedure sets the default value for an attribute. For a state attribute type, it must be set to a value equal to the value of one of the current available states.

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>attrId</td>
<td>The ID of the attribute to set the default for.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>defaultValue</td>
<td>The value to use as the default.</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>updateValues</td>
<td>0 - Do not update attributes</td>
<td>Integer</td>
<td>yes</td>
</tr>
<tr>
<td></td>
<td>1 - Update Attributes that are currently at the old default to the new default</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 - Update all attributes to the new default (default is 0).</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

dp_getDefaultValue
This procedure gets the default value for an attribute. For a state attribute type, it will be a value equal to the value of one of the current available states.

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>attrId</td>
<td>The ID of the attribute to set the default for.</td>
<td>Integer</td>
</tr>
</tbody>
</table>

Result Set

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEFAULT_VALUE</td>
<td>The default value for the given attribute.</td>
<td>String</td>
</tr>
</tbody>
</table>
dp_editAttribute

This procedure will change the information for an attribute.

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>attrId</td>
<td>The ID of the attribute to modify.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>attrName</td>
<td>The new name for the attribute.</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>dataType</td>
<td>The data type of the attribute. Can be one of:</td>
<td>Character</td>
<td></td>
</tr>
<tr>
<td></td>
<td>BLOB='B'</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>EXTERNAL='E'</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>VALUE='V'</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>STATE='S'</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>NUMBER='N'</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>TABLE='T'</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>DATASOURCE='D'</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>DATA COLUMN='C'</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>KEY COLUMN='K'.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>attrDescr</td>
<td>The description of the attribute.</td>
<td>String</td>
<td>yes</td>
</tr>
<tr>
<td>defaultValue</td>
<td>The default value to use when creating.</td>
<td>String</td>
<td>yes</td>
</tr>
<tr>
<td>updateValues</td>
<td>0 - Do not update attributes</td>
<td>Integer</td>
<td>yes</td>
</tr>
<tr>
<td></td>
<td>1 - Update Attributes that are currently at the old default to the new default</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 - Update all attributes to the new default (default is 0).</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

dp_getAttributeInfo

*User Accessible*

This procedure gets the information for an attribute or lists all the available attributes.

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>attrId</td>
<td>The attribute to retrieve the info for. Default = 0 (list all attributes).</td>
<td>Integer</td>
<td>yes</td>
</tr>
</tbody>
</table>
Result Set

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTR_ID</td>
<td>The ID of the attribute.</td>
<td>Integer</td>
</tr>
<tr>
<td>NAME</td>
<td>The name of the attribute.</td>
<td>String</td>
</tr>
<tr>
<td>DESCR</td>
<td>The description of the attribute.</td>
<td>String</td>
</tr>
<tr>
<td>DATA_TYPE</td>
<td>The data type.</td>
<td>Character</td>
</tr>
</tbody>
</table>

**dp_overrideDefaultValue**
This procedure overrides the default value for an attribute for other attribute groups and object type combinations.

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectTypeld</td>
<td>The object type to override for.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>attributeGroupId</td>
<td>The attribute group to override for.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>attributeId</td>
<td>The attribute to override.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>defaultValue</td>
<td>The new override default value.</td>
<td>String</td>
<td></td>
</tr>
</tbody>
</table>

**dp_removeOverrideValue**
This procedure removes an override from an attribute.

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectTypeld</td>
<td>The object type to remove the override from.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>attributeGroupId</td>
<td>The attribute group to remove the override from.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>attributeId</td>
<td>The attribute to remove the override from.</td>
<td>Integer</td>
<td></td>
</tr>
</tbody>
</table>
### dp_getOverrideValue

This procedure retrieves the overrides for a certain object type and attribute group.

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectTypeId</td>
<td>The object type to get the overrides for.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>attributeGroupId</td>
<td>The attribute group to get the overrides for.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>attributeId</td>
<td>The attribute to get the overrides for.</td>
<td>Integer</td>
<td></td>
</tr>
</tbody>
</table>

**Result Set**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEFAULT_VALUE</td>
<td>The override default value.</td>
<td>String</td>
</tr>
</tbody>
</table>

### dp_addAttrSubscription

This procedure adds a subscriber to an attribute. This will be a module and a method that will be alerted when an attribute instance changes value.

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>attrId</td>
<td>The attribute to add the subscription to.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>moduleId</td>
<td>The module that the method belongs to.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>methodId</td>
<td>The method to add as a subscriber.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>objectTypeId</td>
<td>The object type to start retrieving data at.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>numberOfLevels</td>
<td>The levels to go up and look for the object type (0 = first available object of the given type, 1 = object that the attribute belongs to).</td>
<td>Integer</td>
<td>yes</td>
</tr>
</tbody>
</table>
dp_removeAttrSubscription
This procedure removes a subscriber from the attribute.

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>attrId</td>
<td>The attribute to remove the</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td></td>
<td>subscription from.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>moduleId</td>
<td>The module that the method</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td></td>
<td>belongs to.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>methodId</td>
<td>The method to remove from the</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td></td>
<td>subscriber list.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

dp_getAttrSubscriptionInfo
This procedure retrieves a list of subscribers for the given attribute.

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>attrId</td>
<td>The attribute to get the</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td></td>
<td>subscribers for.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Result Set

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>MODULE_ID</td>
<td>The module that the subscriber belongs to.</td>
<td>Integer</td>
</tr>
<tr>
<td>METHOD_ID</td>
<td>The method that is subscribed.</td>
<td>Integer</td>
</tr>
<tr>
<td>LEVELS_UP</td>
<td>The number of levels up the tree to start.</td>
<td>Integer</td>
</tr>
<tr>
<td>OBJECT_TYPE_ID</td>
<td>The object type to start at if found at the desired level.</td>
<td>Integer</td>
</tr>
</tbody>
</table>
dp_addAttrState

This procedure adds a state to the given attribute. The state holds a value that the attribute can be set to. The attribute must have the data type of state before states can be added.

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>attrId</td>
<td>The ID of the attribute to add a state to.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>stateName</td>
<td>The name of the new state.</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>stateValue</td>
<td>The value of the new state. This cannot be blank.</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>isDefault</td>
<td>Specifies whether this state is the default state for its current attribute type (default = false).</td>
<td>Boolean</td>
<td>yes</td>
</tr>
<tr>
<td>stateDescr</td>
<td>The description of the state.</td>
<td>String</td>
<td>yes</td>
</tr>
</tbody>
</table>

Result Set

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>STATE_ID</td>
<td>The ID of the newly created state.</td>
<td>Integer</td>
</tr>
</tbody>
</table>

dp_editAttrState

This procedure edits the information for a state.

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>attrId</td>
<td>The ID of the attribute that the state belongs to.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>stateId</td>
<td>The ID of the state to modify.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>stateName</td>
<td>The name of the state.</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>stateValue</td>
<td>The value of the new state. This cannot be blank..</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>isDefault</td>
<td>Specifies whether this state is the default state for its current attribute type (default = false).</td>
<td>Boolean</td>
<td>yes</td>
</tr>
<tr>
<td>stateDescr</td>
<td>The description of the state.</td>
<td>String</td>
<td>yes</td>
</tr>
</tbody>
</table>
dp_removeAttrState
This procedure removes a state from the attribute.

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>attrId</td>
<td>The attribute to remove the state from.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>stateId</td>
<td>The state to remove.</td>
<td>Integer</td>
<td></td>
</tr>
</tbody>
</table>

dp_getAttrStateInfo
User Accessible
This procedure retrieves information about a state or lists all of the states for an attribute.

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>attrId</td>
<td>The attribute to get state information for.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>stateId</td>
<td>The state to retrieve information for. Default = 0.</td>
<td>Integer</td>
<td>yes</td>
</tr>
</tbody>
</table>

Result Set

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>STATE_ID</td>
<td>The ID of one of the states for this attribute.</td>
<td>Integer</td>
</tr>
<tr>
<td>NAME</td>
<td>The name of the state.</td>
<td>String</td>
</tr>
<tr>
<td>STATE_VALUE</td>
<td>The value of the state.</td>
<td>String</td>
</tr>
<tr>
<td>IS_DEFAULT</td>
<td>Whether this state is the default for the parent attribute type.</td>
<td>Boolean</td>
</tr>
<tr>
<td>DESCR</td>
<td>The description of the state.</td>
<td>String</td>
</tr>
</tbody>
</table>
dp_getAttrStateInfoByValue

User Accessible

This procedure retrieves information about a state or lists all of the states for an attribute.

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>attrId</td>
<td>The attribute to get state information for.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>stateValue</td>
<td>The value of the state to retrieve information for.</td>
<td>Integer</td>
<td></td>
</tr>
</tbody>
</table>

Result Set

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>STATE_ID</td>
<td>The ID of one of the states for this attribute.</td>
<td>Integer</td>
</tr>
<tr>
<td>NAME</td>
<td>The name of the state.</td>
<td>String</td>
</tr>
<tr>
<td>STATE_VALUE</td>
<td>The value of the state.</td>
<td>String</td>
</tr>
<tr>
<td>IS_DEFAULT</td>
<td>Whether this state is the default for the parent attribute type.</td>
<td>Boolean</td>
</tr>
<tr>
<td>DESCRIPT</td>
<td>The description of the state.</td>
<td>String</td>
</tr>
</tbody>
</table>

dp_addAttrStateSubscription

This procedure adds a subscriber, which consists of a module and a method, to a state. This subscriber will be notified when an attribute is changed to this particular state.

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>attrId</td>
<td>The attribute that the state belongs to.</td>
<td>Integer</td>
<td></td>
</tr>
</tbody>
</table>
dp_removeAttrStateSubscription
This procedure removes a subscriber from a state. The subscriber will no longer be notified when the attribute changes to this state.

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>attrId</td>
<td>The attribute that the state belongs to.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>stateId</td>
<td>The state to remove the subscriber from.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>moduleId</td>
<td>The module that the method belongs to.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>methodId</td>
<td>The method to remove from the subscriber list.</td>
<td>Integer</td>
<td></td>
</tr>
</tbody>
</table>

dp_getAttrStateSubscriptionInfo
This procedure sets the constraints for a numeric attribute.

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>attrId</td>
<td>The attribute that the state belongs to.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>stateId</td>
<td>The state to get the subscribers for.</td>
<td>Integer</td>
<td></td>
</tr>
</tbody>
</table>
Result Set

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>MODULE_ID</td>
<td>The module that is subscribed.</td>
<td>Integer</td>
</tr>
<tr>
<td>METHOD_ID</td>
<td>The method that is subscribed.</td>
<td>Integer</td>
</tr>
<tr>
<td>LEVELS_UP</td>
<td>The number of levels up the tree to start.</td>
<td>Integer</td>
</tr>
<tr>
<td>OBJECT_TYPE_ID</td>
<td>The object type to start at if found at the desired level.</td>
<td>Integer</td>
</tr>
</tbody>
</table>

**dp_setAttributeConstraint**
This procedure sets the constraints for a numeric attribute.

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>attrId</td>
<td>The attribute whose constraints will be set.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>mixValue</td>
<td>Minimum value that the attribute can assume.</td>
<td>Double</td>
<td></td>
</tr>
<tr>
<td>maxValue</td>
<td>Maximum value that the attribute can assume.</td>
<td>Double</td>
<td></td>
</tr>
<tr>
<td>allowNull</td>
<td>Specifies whether null values are allowed.</td>
<td>Boolean</td>
<td></td>
</tr>
<tr>
<td>integerOnly</td>
<td>Specifies whether only integers are allowed.</td>
<td>Boolean</td>
<td></td>
</tr>
<tr>
<td>outputFormat</td>
<td>The format to output the number.</td>
<td>String</td>
<td></td>
</tr>
</tbody>
</table>

**dp_getAttributeConstraint**
This procedure gets the constraints for a numeric attribute.

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>attrId</td>
<td>The attribute whose constraints will be set.</td>
<td>Integer</td>
<td></td>
</tr>
</tbody>
</table>
### Result Set

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>attrId</td>
<td>The attribute whose constraints will be set.</td>
<td>Integer</td>
</tr>
<tr>
<td>mixValue</td>
<td>Minimum value that the attribute can assume.</td>
<td>Double</td>
</tr>
<tr>
<td>maxValue</td>
<td>Maximum value that the attribute can assume.</td>
<td>Double</td>
</tr>
<tr>
<td>allowNull</td>
<td>Specifies whether null values are allowed.</td>
<td>Boolean</td>
</tr>
<tr>
<td>integerOnly</td>
<td>Specifies whether only integers are allowed.</td>
<td>Boolean</td>
</tr>
<tr>
<td>outputFormat</td>
<td>The format to output the number.</td>
<td>String</td>
</tr>
</tbody>
</table>

### Attribute Group Subsystem

**dp_addAttributeGroup**

This procedure adds a new attribute group.

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>attrGroupName</td>
<td>The name of the new group.</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>attrGroupDescr</td>
<td>The description for the new group.</td>
<td>String</td>
<td>yes</td>
</tr>
<tr>
<td>attrGroupType</td>
<td>The type of the attribute group. Can be 'C' (cached), 'R' (relational mapping), 'N' (non-cached). Default = 'C'.</td>
<td>String</td>
<td>yes</td>
</tr>
</tbody>
</table>

**Result Set**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTR_GROUP_ID</td>
<td>The ID of the newly created attribute group.</td>
<td>Integer</td>
</tr>
</tbody>
</table>
Appendix A: Training Exercises
NetObjects Stored Procedures

dp_removeAttributeGroup
This procedure removes an attribute group.

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>attrGroupId</td>
<td>The attribute group to remove.</td>
<td>Integer</td>
<td></td>
</tr>
</tbody>
</table>

dp_getAttributeGroupInfo
*User Accessible*
This procedure gets the information for an attribute group or lists the available attribute groups.

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>attrGroupId</td>
<td>The ID of the group to get information for. Default = 0 (lists all groups).</td>
<td>Integer</td>
<td>yes</td>
</tr>
</tbody>
</table>

Result Set

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTR_GROUP_ID</td>
<td>The ID of the attribute group.</td>
<td>Integer</td>
</tr>
<tr>
<td>NAME</td>
<td>The name of the attribute group.</td>
<td>String</td>
</tr>
<tr>
<td>DESCR</td>
<td>The description of the attribute group.</td>
<td>String</td>
</tr>
</tbody>
</table>

dp_editAttributeGroup
This procedure changes the information for a given attribute group.

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>attrGroupId</td>
<td>The ID of the attribute group to modify.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>attrGroupName</td>
<td>The name of the attribute group.</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>attrGroupDescr</td>
<td>The description of the attribute group.</td>
<td>String</td>
<td>yes</td>
</tr>
</tbody>
</table>
dp_addAttrToAttrGroup
This procedure adds an attribute to the attribute group.

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>attrGroupId</td>
<td>The attribute group to add to.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>attrId</td>
<td>The attribute to add to the group.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>audit</td>
<td>Whether to audit the attribute in this group. Default = false.</td>
<td>Boolean</td>
<td>yes</td>
</tr>
</tbody>
</table>

dp_removeAttrFromAttrGroup
This procedure removes an attribute from an attribute group.

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>attrGroupId</td>
<td>The group to remove from.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>attrId</td>
<td>The attribute to remove from the group.</td>
<td>Integer</td>
<td></td>
</tr>
</tbody>
</table>

dp_getAttrsInAttrGroup
This procedure lists the attribute that belong to a given attribute group.

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>attrGroupId</td>
<td>The attribute group to get the attributes for.</td>
<td>Integer</td>
<td></td>
</tr>
</tbody>
</table>

Result Set

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTR_ID</td>
<td>An attribute that resides in the group.</td>
<td>Integer</td>
</tr>
<tr>
<td>NAME</td>
<td>The name of the attribute.</td>
<td>String</td>
</tr>
<tr>
<td>AUDIT_ENABLED</td>
<td>Whether audit is enabled for this attribute in this group.</td>
<td>Boolean</td>
</tr>
</tbody>
</table>
dp_enableAttrAuditing
This procedure enables auditing for an attribute in a given group.

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>attrGroupId</td>
<td>The attribute group that the attribute belongs to.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>attrId</td>
<td>The attribute to audit.</td>
<td>Integer</td>
<td></td>
</tr>
</tbody>
</table>

dp_disableAttrAuditing
This procedure disables attribute auditing for an attribute in a given group.

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>attrGroupId</td>
<td>The attribute group that the attribute belongs to.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>attrId</td>
<td>The attribute to disable auditing for.</td>
<td>Integer</td>
<td></td>
</tr>
</tbody>
</table>

dp_addRelAttrToAudit
This procedure adds a related attribute to be audited along with the given attribute.

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>attrGroupId</td>
<td>The attribute group that the attributes belong to.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>parentAttrId</td>
<td>The parent attribute</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>attrId</td>
<td>The attribute to audit at the same time.</td>
<td>Integer</td>
<td></td>
</tr>
</tbody>
</table>
dp_removeRelAttrFromAudit
This procedure will remove an attribute from being audited along with the given attribute.

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>attrGroupId</td>
<td>The group that the attributes belong to.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>parentAttrId</td>
<td>The parent attribute to remove the relation from.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>attrId</td>
<td>The attribute to remove from the relation.</td>
<td>Integer</td>
<td></td>
</tr>
</tbody>
</table>

Result Set

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTR_ID</td>
<td>The ID of an attribute that will be audited along with the parent attribute.</td>
<td>Integer</td>
</tr>
</tbody>
</table>
Appendix A: Training Exercises

NetObjects Stored Procedures

Page 240

Operational Insight R362.1 Programmer’s Guide

dp_addAttrGroupCollAssoc
This procedure will add an attribute group to a collection.

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>attrGroupId</td>
<td>The attribute group to add to the collection.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>collectionId</td>
<td>The collection to add the object type to.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>required</td>
<td>Specifies whether the attribute group is required in the collection. Will be used to determine access to a view. Default = false(0).</td>
<td>Boolean</td>
<td>yes</td>
</tr>
</tbody>
</table>

dp_removeAttrGroupCollAssoc
This procedure will remove an attribute from a collection.

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>attrGroupId</td>
<td>The attribute group to add to the collection.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>collectionId</td>
<td>The collection to add the object type to.</td>
<td>Integer</td>
<td></td>
</tr>
</tbody>
</table>

dp_getAttrGroupCollAssoc
This procedure will return a list of the collections that this attribute group belongs to.

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>attrGroupId</td>
<td>The attribute group to add to the collection.</td>
<td>Integer</td>
<td></td>
</tr>
</tbody>
</table>
Result Set

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>COLLECTION_ID</td>
<td>The ID of a collection that this attribute group belongs to.</td>
<td>Integer</td>
</tr>
<tr>
<td>REQUIRED</td>
<td>Whether this attribute group is required for the collection. Will be used to determine access to a view.</td>
<td>Boolean</td>
</tr>
</tbody>
</table>

Object Type Subsystem

dp_addObjectType
This procedure will add an object type to NetObjects.

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectTypeName</td>
<td>The name of the object type.</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>inheritedFrom</td>
<td>The ID of the object type that this one is inherited from.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>isTreeEnabled</td>
<td>Whether this object type should be displayed on the tree.</td>
<td>Boolean</td>
<td></td>
</tr>
<tr>
<td>imageIndex</td>
<td>The ID of the image to display for this object type.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>objectTypeDescription</td>
<td>The description for this object type.</td>
<td>String</td>
<td>yes</td>
</tr>
</tbody>
</table>

Result Set

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>OBJECT_TYPE_ID</td>
<td>The ID of the newly created object type.</td>
<td>Integer</td>
</tr>
</tbody>
</table>
Appendix A: Training Exercises
NetObjects Stored Procedures

dp_removeObjectType
This procedure will remove an object type from NetObjects.

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectTypeld</td>
<td>The ID of the object type to remove.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>recursiveObjectRemoval</td>
<td>All objects of the given type will be removed. This flag specifies if everything that belongs to the deleted objects should be deleted as well. For example, if a building is deleted, you would want to delete all the rooms it contains. Default = false.</td>
<td>Boolean</td>
<td>yes</td>
</tr>
</tbody>
</table>

dp_editObjectType
This procedure will change the information for a given object type. Note that the inherited type cannot be changed once the object type has been created.

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectTypeld</td>
<td>The ID of the object type to edit.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>objectTypeld</td>
<td>The new name of the object type.</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>isTreeEnabled</td>
<td>Whether the object type will be seen on the tree.</td>
<td>Boolean</td>
<td></td>
</tr>
<tr>
<td>imageIndex</td>
<td>The image to use for this object type.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>objectTypeDescription</td>
<td>The description of the object type.</td>
<td>String</td>
<td>yes</td>
</tr>
</tbody>
</table>
dp_getObjectTypeInfo

*User Accessible*

This procedure will retrieve the information for a given object type.

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectTypeld</td>
<td>The ID of the object type to get info for. Default = 0 (lists all object types).</td>
<td>Integer</td>
<td>yes</td>
</tr>
</tbody>
</table>

**Result Set**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>OBJECT_TYPE_ID</td>
<td>The ID of the object type.</td>
<td>Integer</td>
</tr>
<tr>
<td>NAME</td>
<td>The name of the object type.</td>
<td>String</td>
</tr>
<tr>
<td>DESCR</td>
<td>The description of the object type.</td>
<td>String</td>
</tr>
<tr>
<td>INHERITED_FROM_ID</td>
<td>The ID of the object type that this one is inherited from.</td>
<td>Integer</td>
</tr>
<tr>
<td>TREE_ENABLED</td>
<td>Whether this object type is tree enabled.</td>
<td>Boolean</td>
</tr>
<tr>
<td>IMAGE_ID</td>
<td>The ID of the image to use when displaying this object type.</td>
<td>Integer</td>
</tr>
</tbody>
</table>

dp_addObjectTypeToRelation

This procedure will relate two object types. When object instances are created, related objects can be automatically be created as well. One thing to note is that object types can only be related using strong relationships.

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>parentObjTypeld</td>
<td>The object type to add the related object to.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>childObjTypeld</td>
<td>The object type that will be related.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>relationshipTypeld</td>
<td>The relationship to use.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>updateObjects</td>
<td>Whether to update all objects in the field with this new relationship.</td>
<td></td>
<td>yes</td>
</tr>
</tbody>
</table>
dp_removeObjectTypeFromRelation
This procedure will break the relationship between two object types. This relationship will be broken for all existing objects of the given type. As well, new ones will not have the related object created for them.

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>parentObjTypeId</td>
<td>The parent to remove the relationship from (0 = apply to all objects).</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>childObjTypeId</td>
<td>The child to remove from the relationship. (0 = apply to all child objects).</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>relationshipTypeId</td>
<td>The relationship to remove the object from (0 = all relationships).</td>
<td>Integer</td>
<td></td>
</tr>
</tbody>
</table>

dp_getObjectTypesByRelation
This procedure will retrieve the object types that are related a certain way.

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>parentObjectTypeId</td>
<td>The parent object to start with. Default = 0 (all objects).</td>
<td>Integer</td>
<td>yes</td>
</tr>
<tr>
<td>relationshipTypeId</td>
<td>The relationship to retrieve. Default = 0 (all relationships).</td>
<td>Integer</td>
<td>yes</td>
</tr>
<tr>
<td>recursive</td>
<td>Whether to recursively get all the related objects. Default = false.</td>
<td>Boolean</td>
<td>yes</td>
</tr>
</tbody>
</table>
Result Set

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>PARENT_OBJECT_TYPE_ID</td>
<td>The ID of the parent object type.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>FROM_INHERITED_TYPE</td>
<td>Specifies whether this came from a type that this object type inherited from.</td>
<td>Boolean</td>
<td></td>
</tr>
<tr>
<td>RELATIONSHIP_TYPE_ID</td>
<td>The relationship to the parent.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>OBJECT_TYPE_ID</td>
<td>The ID of the object type.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>NAME</td>
<td>The name of the object type.</td>
<td>Integer</td>
<td></td>
</tr>
</tbody>
</table>

dp_getObjectTypeRelationTree
This procedure will retrieve the object types that are related a certain way and return it formatted for the Java Tree.

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>parentObjectTypeId</td>
<td>The parent object to start with. Default = 0 (all objects).</td>
<td>Integer</td>
<td>yes</td>
</tr>
<tr>
<td>relationshipTypeId</td>
<td>The relationship to retrieve. Default = 0 (all relationships).</td>
<td>Integer</td>
<td>yes</td>
</tr>
<tr>
<td>recursive</td>
<td>Whether to recursively get all the related objects. Default = false.</td>
<td>Boolean</td>
<td>yes</td>
</tr>
<tr>
<td>collapsed</td>
<td>Whether the tree should be returned collapsed. Default = true.</td>
<td>Boolean</td>
<td>yes</td>
</tr>
</tbody>
</table>
Appendix A: Training Exercises
NetObjects Stored Procedures

Result Set

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>NodeLevel</td>
<td>The level of the node in the tree.</td>
<td>Integer</td>
</tr>
<tr>
<td>NodeText</td>
<td>The text to display in the tree.</td>
<td>String</td>
</tr>
<tr>
<td>ImageIndex</td>
<td>The index of the image to display (corresponds to the images in dp_getImageInfo).</td>
<td>Integer</td>
</tr>
<tr>
<td>Collapsed</td>
<td>Specifies whether the node is collapsed or expanded.</td>
<td>Boolean</td>
</tr>
<tr>
<td>NodeData</td>
<td>The data that resides on the tree.</td>
<td>Hash Map (OBJECT_TYPE_ID, FIT, REL_ID)</td>
</tr>
</tbody>
</table>

**dp_addObjTypeAttrGroupAssoc**

This procedure will associate an attribute group to the object type. When object instances are created, the attribute groups will automatically be created as well. This updates all existing object instances.

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectTypeld</td>
<td>The object type to add an association to.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>attrGroupld</td>
<td>The attribute group to associate.</td>
<td>Integer</td>
<td></td>
</tr>
</tbody>
</table>

**dp_removeObjTypeAttrGroupAssoc**

This procedure will remove an attribute group from being associated with an object type. Object instances will automatically be updated.

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectTypeld</td>
<td>The object type to remove the association from.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>attrGroupld</td>
<td>The attribute group to remove from the association.</td>
<td>Integer</td>
<td></td>
</tr>
</tbody>
</table>
dp_getObjTypeAttrGroupAssoc
This procedure will retrieve the attribute groups that are associated with a given object type.

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectTypId</td>
<td>The object type to get the associations for.</td>
<td>Integer</td>
<td></td>
</tr>
</tbody>
</table>

Result Set

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTR_GROUP_ID</td>
<td>The ID of the attribute group.</td>
<td>Integer</td>
</tr>
<tr>
<td>NAME</td>
<td>The name of the attribute group.</td>
<td>String</td>
</tr>
<tr>
<td>DESCR</td>
<td>The description of the attribute group.</td>
<td>String</td>
</tr>
<tr>
<td>FROM_INHERITED_TYPE</td>
<td>Specifies whether this attribute group was from a parent type.</td>
<td>Boolean</td>
</tr>
</tbody>
</table>

dp_addObjTypeViewAssoc
This procedure will associate a view with an object type. This will cause the view to appear in the tree for any user that has access to an object of the given type as well as access to at least one of the necessary attribute groups.

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectTypId</td>
<td>The object type to add the view to.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>viewId</td>
<td>The ID of the view to add.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>isDefaultView</td>
<td>Specifies whether this is the default view for this object type.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
dp_removeObjTypeViewAssoc
This procedure removes a view from being associated with the given object type.

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectTypeId</td>
<td>The object type to remove the view from.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>viewId</td>
<td>The view ID to remove from the object type.</td>
<td>Integer</td>
<td></td>
</tr>
</tbody>
</table>

dp_setObjTypeDefaultView
This procedure will set which of the available views will be the default for the given object type. This will allow the tree to know what to use as the link for the object in the tree. All other views can then be displayed below the object.

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectTypeId</td>
<td>The object type to set the default for.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>viewId</td>
<td>The view ID to make the default.</td>
<td>Integer</td>
<td></td>
</tr>
</tbody>
</table>

dp_clearObjTypeDefaultView
This procedure will clear the default view for a given object type. This will not delete the view that is currently the default. Instead, it will simply remove it from being the default.

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectTypeId</td>
<td>The object type to remove the default from.</td>
<td>Integer</td>
<td></td>
</tr>
</tbody>
</table>

dp_getObjTypeViewAssoc
This procedure will list the views that are associated with the given object type.

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectTypeId</td>
<td>The object type to get the views for.</td>
<td>Integer</td>
<td></td>
</tr>
</tbody>
</table>
**dp_addObjectTypeCollAssoc**  
This procedure allows an object type to be added to a collection.  
**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectTypeld</td>
<td>The object type to add to the collection.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>collectionId</td>
<td>The collection to add the object type to.</td>
<td>Integer</td>
<td></td>
</tr>
</tbody>
</table>

**dp_removeObjectTypeCollAssoc**  
This procedure allows an object type to be removed from a collection.  
**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectTypeld</td>
<td>The object type to remove from the collection.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>collectionId</td>
<td>The collection to remove the object type from.</td>
<td>Integer</td>
<td></td>
</tr>
</tbody>
</table>

**dp_getObjectTypeCollAssoc**  
This procedure will return the collections that an object type is associated with.  
**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectTypeld</td>
<td>The object type to get the associated collections from.</td>
<td>Integer</td>
<td></td>
</tr>
</tbody>
</table>
Appendix A: Training Exercises
NetObjects Stored Procedures

Result Set

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>COLLECTION_ID</td>
<td>The ID of an associated collection.</td>
<td>integer</td>
</tr>
</tbody>
</table>

Object Subsystem

dp_addObject
This procedure adds an object instance and possibly a number of related objects.

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectTypeId</td>
<td>The ID of the object type to implement.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>objectName</td>
<td>The name of the object to be created. Default = name of its object type.</td>
<td>String</td>
<td>yes</td>
</tr>
<tr>
<td>objectDescription</td>
<td>The description of the object to be added.</td>
<td>String</td>
<td>yes</td>
</tr>
<tr>
<td>recursive</td>
<td>Specifies whether instances of all related object types should be created as well. Default = false.</td>
<td>Boolean</td>
<td>yes</td>
</tr>
</tbody>
</table>

Result Set

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>OBJECT_ID</td>
<td>The ID of the first object to be created.</td>
<td>Integer</td>
</tr>
</tbody>
</table>

dp_removeObject
This procedure removes an object instance and possibly all the object instances that belong to the specified object.

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>The ID of the object to remove.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>recursive</td>
<td>Specifies whether to recursively remove all strongly related objects. Default = false.</td>
<td>Boolean</td>
<td>yes</td>
</tr>
</tbody>
</table>
dp_getObjectInfo

User Accessible
This procedure retrieves the information for a given object or lists all of the objects.

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>The ID of the object to retrieve. Default = 0 (get all object Info).</td>
<td>Integer</td>
<td>yes</td>
</tr>
</tbody>
</table>

Result Set

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>OBJECT_ID</td>
<td>The ID of the object.</td>
<td>Integer</td>
</tr>
<tr>
<td>NAME</td>
<td>The name of the object.</td>
<td>String</td>
</tr>
<tr>
<td>DESCR</td>
<td>The description of the object.</td>
<td>String</td>
</tr>
<tr>
<td>OBJECT_TYPE_ID</td>
<td>The type that this object is.</td>
<td>Integer</td>
</tr>
<tr>
<td>OBJECT_TYPE_NAME</td>
<td>The name of the object type.</td>
<td>String</td>
</tr>
</tbody>
</table>

dp_editObjectInfo

This procedure changes the information for a given object.

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>The ID of the object to edit.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>objectName</td>
<td>The name of the object.</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>objectDescription</td>
<td>The description of the object.</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>yes</td>
</tr>
</tbody>
</table>

dp_listObjectAttrGroups

List the available attribute groups for a given object.

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>The ID of the object to retrieve the groups for.</td>
<td>Integer</td>
<td></td>
</tr>
</tbody>
</table>
Result Set

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTR_GROUP_ID</td>
<td>The ID of the attribute group.</td>
<td>Integer</td>
</tr>
<tr>
<td>NAME</td>
<td>The name of the attribute group.</td>
<td>String</td>
</tr>
<tr>
<td>DESCR</td>
<td>The description of the attribute group.</td>
<td>String</td>
</tr>
</tbody>
</table>

**dp_addObjectToRelation**

This procedure will relate two objects.

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>parentObjectId</td>
<td>The parent object to use in the relation.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>relatedObjectId</td>
<td>The object to relate to the parent.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>relationshipTypeId</td>
<td>The way in which to relate the objects.</td>
<td>Integer</td>
<td></td>
</tr>
</tbody>
</table>

**dp_removeObjectFromRelation**

This procedure breaks the relationship between two objects.

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>parentObjectId</td>
<td>The parent in the relationship.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>relatedObjectId</td>
<td>The object to remove from the relationship.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>relationshipTypeId</td>
<td>The relationship to remove the object from.</td>
<td>Integer</td>
<td></td>
</tr>
</tbody>
</table>
dp_getObjectsByRelation
This procedure gets the objects that are related to a given parent object. This can be done recursively and based on the strength of the relationship.

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>parentObjectId</td>
<td>The object to start with. Default = 0 (get all objects).</td>
<td>Integer</td>
<td>yes</td>
</tr>
<tr>
<td>relationshipTypeId</td>
<td>The relationship to retrieve objects for. Default = 0 (all relationships).</td>
<td>Integer</td>
<td>yes</td>
</tr>
<tr>
<td>recursive</td>
<td>Whether to retrieve recursively. Default = false.</td>
<td>Boolean</td>
<td>yes</td>
</tr>
<tr>
<td>strength</td>
<td>The strength to filter the results by.</td>
<td>Integer (See Relationship Types)</td>
<td>yes</td>
</tr>
</tbody>
</table>

Result Set

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>PARENT_OBJECT_ID</td>
<td>The parent object.</td>
<td>Integer</td>
</tr>
<tr>
<td>RELATIONSHIP_TYPE_ID</td>
<td>The way in which these objects are related.</td>
<td>Integer</td>
</tr>
<tr>
<td>OBJECT_ID</td>
<td>The ID for this object.</td>
<td>Integer</td>
</tr>
<tr>
<td>NAME</td>
<td>The name of the object.</td>
<td>String</td>
</tr>
</tbody>
</table>
dp_getObjectRelationTree

This procedure gets the objects that are related to a given parent object. This can be done recursively and based on the strength of the relationship. The data is returned in a format required for the Java Tree.

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>parentObjectId</td>
<td>The object to start with. Default = 0 (get all objects).</td>
<td>Integer</td>
<td>yes</td>
</tr>
<tr>
<td>relationshipTypeId</td>
<td>The relationship to retrieve objects for. Default = 0 (all relationships).</td>
<td>Integer</td>
<td>yes</td>
</tr>
<tr>
<td>recursive</td>
<td>Whether to retrieve recursively. Default = false.</td>
<td>Boolean</td>
<td>yes</td>
</tr>
<tr>
<td>collapsed</td>
<td>Specifies whether the tree should come back collapsed or not. Default = true.</td>
<td>Boolean</td>
<td>yes</td>
</tr>
<tr>
<td>strength</td>
<td>The strength to filter the results by. Default = 0 (all Strengths or no filter).</td>
<td>Integer (See Relationship Types)</td>
<td>yes</td>
</tr>
</tbody>
</table>

Result Set

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>NodeLevel</td>
<td>The level of the node in the tree.</td>
<td>Integer</td>
</tr>
<tr>
<td>NodeText</td>
<td>The test to display in the tree.</td>
<td>String</td>
</tr>
<tr>
<td>ImageIndex</td>
<td>The index of the image to display (corresponds to the images in dp_getImageInfo).</td>
<td>Integer</td>
</tr>
<tr>
<td>Collapsed</td>
<td>Specifies whether the node is collapsed or expanded.</td>
<td>Boolean</td>
</tr>
<tr>
<td>NodeData</td>
<td>The data that resides on the tree. Contains the object ID and the relationship type ID.</td>
<td>Hash Map (OBJECT_ID, REL_ID)</td>
</tr>
</tbody>
</table>
dp_getObjectTree

*User Accessible*

This procedure retrieves the object tree, which can be used to display the hierarchy to the user.

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>relationshipTypeId</td>
<td>The relationship to get the tree for. Default = 0 (all relationships).</td>
<td>Integer</td>
<td>yes</td>
</tr>
<tr>
<td>includeDefaultViews</td>
<td>Whether to return the default views. Will appear as part of the node data. Default = false.</td>
<td>Boolean</td>
<td>yes</td>
</tr>
<tr>
<td>includeAdditionalViews</td>
<td>Specifies whether to include the rest of the views as well. Default = false.</td>
<td>Boolean</td>
<td>yes</td>
</tr>
<tr>
<td>collapsed</td>
<td>Whether the tree is fully collapsed or not. Default = true.</td>
<td>Boolean</td>
<td>yes</td>
</tr>
<tr>
<td>userId</td>
<td>Specifies the user tree to return. This only applies for administrators so that the tree can be previewed. Non-Administrators can only retrieve their own tree and this parameter does not apply. Default = 0 (currently logged in user).</td>
<td>Integer</td>
<td>yes</td>
</tr>
<tr>
<td>objectId</td>
<td>The object to start at in the tree. Default = 0 (start at the top).</td>
<td>Integer</td>
<td>yes</td>
</tr>
<tr>
<td>overrideTreeEnabled</td>
<td>This allows objects that are not tree enabled to be returned as part of the tree anyway. This will include any associated views as well. Default = false.</td>
<td>Boolean</td>
<td>yes</td>
</tr>
</tbody>
</table>
Result Set

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>NodeLevel</td>
<td>The level of the node in the tree.</td>
<td>Integer</td>
</tr>
<tr>
<td>NodeText</td>
<td>The text to display in the tree.</td>
<td>String</td>
</tr>
<tr>
<td>ImageIndex</td>
<td>The index of the image to display (corresponds to the images in dp_getImageInfo).</td>
<td>Integer</td>
</tr>
<tr>
<td>Collapsed</td>
<td>Specifies whether the node is collapsed or expanded.</td>
<td>Boolean</td>
</tr>
<tr>
<td>NodeData</td>
<td>The data that resides on the tree. Contains the object ID and the relationship type ID.</td>
<td>HashMap (OBJECT_ID, VIEW_ID, REL_ID, PAGE)</td>
</tr>
</tbody>
</table>

**dp_setAttrGroupPerms**
This procedure will set the attribute group permissions.

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>The ID of the object that the attribute group belongs to.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>attrGroupId</td>
<td>The ID of the attribute group to add perms to (0 = all attribute groups for this object).</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>userId</td>
<td>The user to give permission to.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>userPerms</td>
<td>The level of permission to give to the user. (NO_PERM=0,READ=1,WRITE=2, ADMIN=3)</td>
<td>Integer</td>
<td></td>
</tr>
</tbody>
</table>
dp_removeAttrGroupPerms
This procedure will remove a user from having permission to an attribute group.

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>The object that the attribute group belongs to. Default = 0 (all objects).</td>
<td>Integer</td>
<td>yes</td>
</tr>
<tr>
<td>attrGroupId</td>
<td>The attribute group to remove user permissions from. Default = 0 (all attribute groups for this object).</td>
<td>Integer</td>
<td>yes</td>
</tr>
<tr>
<td>userId</td>
<td>The user to remove permissions for. Default = 0 (all users for this attribute group).</td>
<td>Integer</td>
<td>yes</td>
</tr>
</tbody>
</table>

dp_getAttrGroupPerms
This procedure retrieves the permissions for an attribute group and a given user.

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>The object to retrieve perms for.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>attrGroupId</td>
<td>The attribute group to get perms for.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>userId</td>
<td>The user to get perms for. Default = 0 (All users that have permission to this attribute group).</td>
<td>Integer</td>
<td>yes</td>
</tr>
</tbody>
</table>

Result Set

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>USER_ID</td>
<td>The user that this permission level applies to.</td>
<td>Integer</td>
</tr>
<tr>
<td>USERNAME</td>
<td>The name of this user.</td>
<td>String</td>
</tr>
<tr>
<td>PERM_LEVEL</td>
<td>The permission level that the user has to this attribute group. (NO_PERM = 0, READ = 1, WRITE = 2, ADMIN = 3).</td>
<td>Integer</td>
</tr>
</tbody>
</table>
dp_setObjectUserPerms
This procedure sets the permissions on object.

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>The object to set user permissions on.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>userId</td>
<td>The user to give permissions to.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>permLevel</td>
<td>The level of permission to give the user.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>applyToData</td>
<td>Specifies whether the permissions should be propagated to the attribute groups that belong to this object. Default = false.</td>
<td>Boolean</td>
<td>yes</td>
</tr>
<tr>
<td>recursive</td>
<td>Specifies whether the permissions should be given to all strongly related objects as well. Default = false.</td>
<td>Boolean</td>
<td>yes</td>
</tr>
</tbody>
</table>

dp_getObjectUserPerms
This procedure retrieves the user permissions for an object and a given user.

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>The ID of the object to get perms for.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>userId</td>
<td>The user to retrieve the permissions for. Default = 0 (all users for this object).</td>
<td>Integer</td>
<td>yes</td>
</tr>
</tbody>
</table>

Result Set

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>USER_ID</td>
<td>The ID of the user.</td>
<td>Integer</td>
</tr>
<tr>
<td>USERNAME</td>
<td>The name of this user.</td>
<td>Integer</td>
</tr>
<tr>
<td>PERM_LEVEL</td>
<td>This users level of permission to the object. . (NO_PERM=0, READ=1, WRITE=2, ADMIN=3).</td>
<td>Integer</td>
</tr>
</tbody>
</table>
dp_removeObjectUserPerms
This procedure removes a user from the permissions of an object.

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>The object to remove permissions for.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>userId</td>
<td>The user to remove object permissions for.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>recursive</td>
<td>Whether to remove the permissions to objects that belong to this object. Default = false.</td>
<td>Boolean</td>
<td>yes</td>
</tr>
</tbody>
</table>

dp_getAllData
User Accessible
This procedure retrieves the objects that the user has access to as well as the attributes that they have access to.

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>viewId</td>
<td>The view to retrieve the data for. Default = 0 (get data for all views).</td>
<td>Integer</td>
<td>yes</td>
</tr>
<tr>
<td>objectId</td>
<td>The object to start with. Default = 0 (get data for all objects).</td>
<td>Integer</td>
<td>yes</td>
</tr>
<tr>
<td>recursive</td>
<td>Whether to get data for child objects recursively. Default = false (get only the requested object).</td>
<td>Boolean</td>
<td>yes</td>
</tr>
<tr>
<td>relationshipTypeld</td>
<td>The relationship to get data for. Default = 0 (get data for all relationships). Note that this only applies if getting data recursively.</td>
<td>Integer</td>
<td>yes</td>
</tr>
</tbody>
</table>
## Result Set

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>OBJECT_ID</td>
<td>The ID of the object.</td>
<td>Integer</td>
</tr>
<tr>
<td>NAME</td>
<td>The name of this object.</td>
<td>String</td>
</tr>
<tr>
<td>OBJECT_TYPE_NAME</td>
<td>The name of the object type.</td>
<td>String</td>
</tr>
<tr>
<td>PARENT_OBJECT_ID</td>
<td>The ID of the parent object ID (0 = no parent).</td>
<td>Integer</td>
</tr>
<tr>
<td>RELATIONSHIP_TYPE_ID</td>
<td>The ID of the Relationship to the parent object (0 = no relationship).</td>
<td>Integer</td>
</tr>
<tr>
<td>ATTR_GROUP_ID</td>
<td>The ID of the attribute group.</td>
<td>Integer</td>
</tr>
<tr>
<td>ATTR_GROUP_NAME</td>
<td>The name of the attribute group.</td>
<td>String</td>
</tr>
<tr>
<td>ATTR_ID</td>
<td>The ID of the attribute.</td>
<td>Integer</td>
</tr>
<tr>
<td>ATTR_NAME</td>
<td>The name of the attribute.</td>
<td>String</td>
</tr>
<tr>
<td>DATA_TYPE</td>
<td>The data type of the attribute.</td>
<td>Character (See Attribute)</td>
</tr>
<tr>
<td>ATTR_VALUE</td>
<td>The actual attribute value.</td>
<td>String</td>
</tr>
</tbody>
</table>

### dp_getAttributeData

**User Accessible**

This procedure gets the value for a given attribute, group of attributes, or all attributes for a given object.

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>The ID of the object that the attribute belongs to.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>attrGroupId</td>
<td>The ID of the attribute group that the attribute belongs to.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>attrId</td>
<td>The ID of the attribute to get data for.</td>
<td>Integer</td>
<td></td>
</tr>
</tbody>
</table>
Result Set

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
<td>The name of the attribute.</td>
<td>String</td>
</tr>
<tr>
<td>DATA_TYPE</td>
<td>The data type of the attribute.</td>
<td>Character (See Attribute)</td>
</tr>
<tr>
<td>ATTR_VALUE</td>
<td>The value of the attribute.</td>
<td>String</td>
</tr>
</tbody>
</table>

**dp_setAttributeData**

*User Accessible*

This procedure sets the value of an attribute for a given object and group.

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>The ID of the object that the attribute belongs to.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>attrGroupId</td>
<td>The ID of the attribute group that the attribute belongs to.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>attrId</td>
<td>The attribute to set the data for.</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>attrValue</td>
<td>The new value for the attribute.</td>
<td>String</td>
<td></td>
</tr>
</tbody>
</table>

**User Accessible Database Views**

The following database views are exposed through the NetObjects data source driver and are available to all logins of NetObjects. Users may select from the available views, and NetObjects will automatically apply permissions to the query to ensure that the user only has access to the objects and attributes that they have been granted permission to.
### Objects

#### 3.1.1 V_OBJECTS
This view exposes the objects that a user has access to.

**Available Columns**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>OBJECT_ID</td>
<td>The ID of the object.</td>
<td>Integer</td>
</tr>
<tr>
<td>NAME</td>
<td>The name of the object.</td>
<td>String</td>
</tr>
<tr>
<td>DESCR</td>
<td>The description of the object.</td>
<td>String</td>
</tr>
<tr>
<td>OBJECT_TYPE_ID</td>
<td>The ID for the object's type.</td>
<td>Integer</td>
</tr>
<tr>
<td>OBJECT_TYPE_NAME</td>
<td>The name of the object's type.</td>
<td>String</td>
</tr>
<tr>
<td>OBJECT_TYPE_DESCR</td>
<td>The description of the object's type.</td>
<td>String</td>
</tr>
</tbody>
</table>

#### 3.1.2 V_OBJECT_VALUES
This view exposes the objects that a user has access to.

**Available Columns**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>OBJECT_ID</td>
<td>The ID of the object.</td>
<td>Integer</td>
</tr>
<tr>
<td>NAME</td>
<td>The name of the object.</td>
<td>String</td>
</tr>
<tr>
<td>OBJECT_TYPE_ID</td>
<td>The ID for the object's type.</td>
<td>Integer</td>
</tr>
<tr>
<td>OBJECT_TYPE_NAME</td>
<td>The name of the object's type.</td>
<td>String</td>
</tr>
<tr>
<td>ATTR_GROUP_ID</td>
<td>The ID of the attribute group.</td>
<td>Integer</td>
</tr>
<tr>
<td>ATTR_GROUP_NAME</td>
<td>The name of the attribute group.</td>
<td>String</td>
</tr>
<tr>
<td>ATTR_ID</td>
<td>The ID of an attribute in the group.</td>
<td>Integer</td>
</tr>
<tr>
<td>ATTR_NAME</td>
<td>The name of the attribute.</td>
<td>String</td>
</tr>
<tr>
<td>DATA_TYPE</td>
<td>The data type of the attribute.</td>
<td>Character</td>
</tr>
<tr>
<td>ATTR_VALUE</td>
<td>The attribute's value</td>
<td>String</td>
</tr>
<tr>
<td>STATE_VALUE</td>
<td>The value if it is a state.</td>
<td>String</td>
</tr>
</tbody>
</table>
**V_OBJECT_RELATIONSHIPS**

This view shows the relationships of objects. The user must have access to the child and parent object in order to query a given relationship.

**Available Columns**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>OBJECT_ID</td>
<td>The ID of the object.</td>
<td>Integer</td>
</tr>
<tr>
<td>NAME</td>
<td>The name of the object.</td>
<td>String</td>
</tr>
<tr>
<td>RELATIONSHIP_TYPE_ID</td>
<td>The relationship to the object's parent.</td>
<td>Integer</td>
</tr>
<tr>
<td>PARENT_ID</td>
<td>The parent's ID.</td>
<td>Integer</td>
</tr>
<tr>
<td>PARENT_NAME</td>
<td>The parent's name.</td>
<td>String</td>
</tr>
</tbody>
</table>

**V_AUDIT_TRAIL**

This view exposes the audit history. You must have access to both the attribute group and the object in order to get access to audit history for a given attribute value.

**Available Columns**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRANS_NO</td>
<td>The transaction number, which is used to group attributes that change at the same time.</td>
<td>Integer</td>
</tr>
<tr>
<td>ATTR_NAME</td>
<td>The name of the attribute that changed.</td>
<td>String</td>
</tr>
<tr>
<td>DATE_INSERTED</td>
<td>The date that the change occurred.</td>
<td>Integer</td>
</tr>
<tr>
<td>ATTR_VALUE</td>
<td>The value of the attribute before it changed.</td>
<td>String</td>
</tr>
<tr>
<td>USER_ID</td>
<td>The ID of the user that changed the attribute.</td>
<td>Integer</td>
</tr>
<tr>
<td>USERNAME</td>
<td>The name of the user that changed the attribute.</td>
<td>String</td>
</tr>
<tr>
<td>ATTR_ID</td>
<td>The ID of the attribute that changed.</td>
<td>Integer</td>
</tr>
<tr>
<td>OBJECT_ID</td>
<td>The ID of the object it belongs to.</td>
<td>Integer</td>
</tr>
<tr>
<td>ATTR_GROUP_ID</td>
<td>The ID of the attribute group that it belongs to.</td>
<td>Integer</td>
</tr>
<tr>
<td>ATTR_GROUP_NAME</td>
<td>The name of the attribute group that it belongs to.</td>
<td>String</td>
</tr>
</tbody>
</table>
Appendix A: Training Exercises

NetObjects Stored Procedures
Index

A
abstracting data sources, 36
  using SQL to retrieve tag data, 36
  using tag-based commands to get relational data, 37
Add, 111
addItem, 137
addQuery, 129, 130
addRow, 145
addSeries, 106, 107
addSeriesFromClass, 114, 117
AddSQL, 56
AddXYStatus, 108, 109
ADO, 28, 157
  Excel, 157
    using a Recordset object, 157
    using MSXML, 158
    WriteVariantBlob, 153
advanced data access commands, 33–36
  processedTagHistory, 35
  rawTagHistory, 34
advanced HTML elements, 11–12
advanced relational database metadata, 41–51
  columnPrivileges function, 50
    columns function, 49
    foreignKeys function, 50
    getInfo function, 41, 42
    getTypeInfo function, 51
    primaryKeys function, 50
    procedureColumns function, 50
    procedures function, 50
    statistics function, 50
    tables function, 49
advanced tag-based metadata, 52–53
  getSupportedAttributes function, 52
  tagAttributes function, 52
  tagSearch function, 52
aggregate, 35
AlertDialog, 75, 76, 79
AllowAxesMove, 105
AllowRightClickUnzoom, 104
applet element, 11, 95–102
  calendar, 133
    menu, 131–133
    NetTrend, 103
    WebDataSet, 95–102
applyTime, 135
AsDate, 99
AsFloat, 99
AsInteger, 99
AsString, 99
AsTimeString, 99
attribute##, 53
autoCreateAxes, 136
autosize feature, 183
Average, 111
axAxis, 125
B
bActive, 107
basic HTML elements, 7–9
bAsyncynchronous, 130
bgColor, 132
BLOB, 152
blobData, 29
bModeless, 76
bNewWin, 81
bNoColumnRestriction, 85
bNoLoad, 82
bNoQueryOnBegin, 129
bNoTrend, 85
bPause, 129
bShowIfBad, 108
bSQL, 85
bSQLOnly, 85
buttonconfig.xml file, 168
bWasRight, 120, 121, 125
C
Calendar object, 133
category, 54
ChangePassword, 56, 59
cBadColor, 108
clear, 145
closeSession, 59
Color, 79, 83
color palette, 83
columnNameMask, 49
columnPrivileges, 41, 50
columns, 41, 49
columns function, 49
columns method, 49
common HTML elements, 9–11
ConfigFiler, 139
ConfigFiler object, 139–140
configID, 63
configID, 103
classifying trends, 105–120
  adding a line series, 106
  adding axes, 118
  adding bars, 117
adding constants, 114
adding functions, 111, 112
PNChartSeries object, 107
PNDataSourceManager object, 110
confirm, 60
ConfirmDialog, 75, 78
confirmSession, 147
crCursor, 126
creating custom calculations, 112
csSeries, 121, 125
Custom, 112
custom toolbars and buttons, NetDraw, 168
D
data access commands, 28–37
  advanced, 33, 36
  SQL-based, 28
  Tag-based, 30
  tag-based historical, 32
  tag-based real-time, 31
database content, 63
dataEncoding, 30
DataHub object, 129
datasource, 152, 154
DateFormat, 104
DateTime, 79
dateToMinutes, 92
dateToSeconds, 92
Debug, 105
deleteRow, 145
deleteSelected, 145
Delimiter, 132
desc, 39
dialog boxes, 75–88
  alert dialog, 76
  color dialog, 83
  confirm dialog, 78
  date and time selection dialog, 88
  ErrorMessage dialog, 78
  ExtendedAlert dialog, 78
  font dialog, 83
  load dialog, 81
  message dialog, 75
  PromptUser dialog, 77
  save dialog, 79
  SQL editor dialog, 87
  tag search dialog, 84
Divide, 111
doHierarchal, 40
doLogin, 147
doLogout, 147
draw, 135, 143
DrawFrame, 104
Dynamic HTML, 12
  scripting, 23
  styles, 12
E
Enable, 137
End, 32, 118
end, 34, 35
enumFontStyle, 84
EoF, 98, 151
ErrorMessage, 75, 78
Excel
  using ADO in, 157
exercise, 193–205
  Accessing Data Using the WebDataSet, 202
  Accessing Non-Tag Data, 203
  Accessing Real-time Data, 202
  Adding Data to a Trend, 204
  Adding Server Data to a Trend, 204
  Basic HTML Page, 193
  Creating a Tabbed Dialog, 204
  Creating an Active Server Page, 205
  HTML and Scripting, 198
  HTML Forms for Data Access, 200
  HTML Forms for Data Entry, 200
  RL Queries for SQL Data, 199
  Table Layout, 194
  URL Queries for Tag Data, 199
  Using DHTML Styles, 194
  Using PNSP for Current Values, 195
  Using PNSP for SQL, 197
  Using PNSP for Tag History, 196
  Using PNSP in Forms, 201
explicitLogin, 58, 60, 62, 63
ExtendedAlertDialog, 75, 78
extending the context menu, 181
F
FieldByName, 98, 100, 151
FieldCount, 98, 151
Fields, 98, 151
First, 98, 151
fnFunc, 130
folderPath, 40
Font, 79, 83
foreignKeys, 41, 50
foreignKeyTableName, 50
Index

format, 29, 31, 32, 61
FormatDateString, 92
FormatFloat, 91
forms
  data access, 69
data entry, 70
Function, 32
function, 35, 39
function0, 56
function99, 56
funcValidation, 77
G
generate scripts, 23
global date/time functions, 92
global functions, 91
global numerical functions, 91, 93
gotoTab, 142
Group Box Control object, 142–143
creating scripts, 23
Dynamic HTML, 12
form objects, 67
generating HTML through PNSP, 14
JavaScript versus VBScript, 23
selection lists, 69
VBScript versus JavaScript, 23
I
ID, 103
ignoreWhere, 30
imgSize, 145
includeSystemTables, 49
Info, 61
infoType, 38, 42
intfType, 39, 52, 53
intToHex, 91
iPaletteNumber, 83
isNull, 99
iStartPathID, 81
J
JavaScript, 23
JWebDataSet, 147
JWebDataSet object, 96
JWebRecordSet object, 98, 151
K
key, 56
L
Last, 98, 151
launchColorWindow, 83
List Control object, 143–146
ListConfig, 139
Load, 81
LoadConfig, 139
Log off, 59
LogonConfig, 57
LogonPage, 57
LogOut, 59
M
master configuration file, NetDraw SDK, 165
defining, 165
loading, 167
  system key/value pairs, 165
Max, 112
maxRecords, 33, 34, 35, 39
Menu object, 131–133
MessageDialog, 75
metadata
access commands, 37
accessing, 100
advanced relational database, 41–51
advanced tag-based, 52–53
basic requests, 38
metadata data access commands
  retrieving a list of functions, 41
  retrieving a list of tables, 41
  retrieving a list of tags, 39, 40
  retrieving data source names, 38
MetaDataSet, 100
Min, 112
minutesToDate, 92
Moving Average, 112
Multiply, 111

N
nClickCount, 121
NetDAC, 149–155
NetDraw SDK, 165
  attach object.htc in run mode, 182
  autosize feature, 183
  custom objects, 176
    changing behavior according to mode, 182
    events, 179
    extending the context menu, 181
    inserting, 178
    preventing grouping, 183
  custom toolbars and buttons, 168
  master configuration file, 165
    examples, 170
  non-visual objects, 181
    writing HTC, 181
  property editors, 174
  property inspector, 173
    defining objects that use a property, 174
    getter/setter code, 173
script, 172
NetPortal
  definition, 2
  Function Summary, 64
  Parameter Summary, 64
  RPC Call Summary, 64
  Session Information, 61
  VersionInfo, 61
NetPortal functions, 53–64
  ChangePassword, 59
  Info, 61
  PingSession, 61
  SubmitLogin, 57
  VersionInfo, 61
NetTrend
  applet element, 103
  applet parameters, 103
new, 60, 61, 63
Next, 98, 151
NextTab, 142
nFolderID, 140
nFontSize, 84
nGCPathID, 140
nIndex, 121
nInterval, 130
nMaxLength, 77
non-visual objects, 181
nRefresh, 107
nStatus, 108
nValues, 85
nX, 132
nXPos, 120, 121, 125
nXPosTrend, 121
nY, 132
nYPos, 120, 121, 125
nYPosTrend, 121

O
objects
  Calendar, 133
  ConfigFiler, 139–140
  DataHub, 129
  Group Box Control, 142–143
  JWebDataSet, 96
  JWebRecordSet, 98, 151
  List Control, 143–146
  Menu, 131–133
  Recordset, 157
  Slider Control, 146–147
  Tabbed Dialog Control, 140–142
  Toolbar, 136–138
  Trend Control Bar, 134–136
  XField, 98
  onAxisClick, 125
  OnClick, 104, 120
  onCursor, 126
  OnDataLoaded, 124
  OnDbClick, 104, 122
  onDeleteCursor, 126
  OnError, 104, 122
  onLegendClick, 124
  onLegendMouseMove, 125
  onMenuClick, 136
OnSeriesClick, 121
onZoom, 123
oParams, 130
OpenCalendarDialog, 88
OpenLoadFileDialog, 81
openLoginDlg, 147
openRecordset, 96, 97, 98
OpenSaveFileDialog, 79
OpenSQLInputDialog, 87
OpenTagListWindow, 84
treeFrame, 80
trend, 85
oWDS, 80, 82, 85, 88
oWebDS, 139
oWin, 76
oWindow, 82, 83, 85, 88
P
packet provider, 159
definition, 2
interface, 159
sample, 161
pageAxis, 135
ParseDateFormat, 105
password, 57, 58, 60, 61, 63
PingSession, 61
PNSP
band element, 15
dataset element, 16
definition, 2
error element, 17
errorTextElement, 17
generating client-side scripts, 72
generating form elements, 71
generating HTML, 14
generating queries from, 74
global element, 18
import element, 18
login element, 19
master-detail bands, 73
permission element, 20
using dynamic properties, 73
using parameters, 22
value element, 21
variable element, 22
Position, 119
PreviousTab, 142
primaryKeys, 42, 50
primaryKeyTableName, 50
Prior, 98, 151
procedureColumns, 42, 50
procedures, 42, 50
procedures, to
use ADO in Excel, 157
processedTagHistory, 33, 35, 36
PromptUser, 75, 77
property editors, 174
property inspector, NetDraw SDK, 173
Q
QueryPath, 103
R
rawTagHistory, 33, 34, 35
RealTimeRetryCount, 103
RecordCount, 98, 151
RecordSet, 61
recordset, 98
redirect, 57, 58
refetch, 136
refresh, 145
reset, 136
rpc, 34, 36, 39
rpc=writeBLOB, 30
S
sampleInterval, 36
Save, 79
SaveConfig, 139, 140
sBackgroundColor, 136
sButtons, 76
sCaptionColor, 136
sCommand, 96
sConfigID, 139
sContent, 140
scripting
accessing HTML objects, 24
creating scripts in HTML, 23
DHTML, 23
sData, 80
sDefault, 77
sDesc, 123
sDescr, 140
sDialogHeader, 80, 82
sDialogTitle, 80, 82
sDS, 85, 88, 130
selectAxis, 135
selectByString, 145
selectRow, 145
sErrFunc, 139
### Session Manager
- sessionIsGood, 147
- setColumns, 145
- setJSFunction, 112
- setPositionPct, 119
- setStart, 118
- setValue, 146
- sEvent, 132
- sExtension, 81
- sFileDesc, 80
- sFileName, 80
- sFontName, 84
- sFunction, 107
- sHelpURL, 76, 77
- sHex, 83
- sHistSrc, 107
- sHistTag, 107
- showLabels, 137
- ShowMenu, 132
- sHTTPQuery, 161
- sIcon, 76, 77, 81
- sItems, 132
- sLabel, 108

### Slider Control object
- 146–147
- sMsg, 123
- sName, 140
- SnapTime, 31
- sPassword, 85
- sPath, 85, 96
- sPrompt, 76, 77
- SQL, 29, 152
- sql, 38

### SQL Editor
- 79, 87
- sQuery, 130
- sRTSrc, 107
- sRTTag, 107
- sServer, 81, 82
- sShortcut, 140
- sSQL, 88
- sStart, 107
- Start, 118
- start, 32, 34, 35, 129
- statistics, 42, 50
- status, 154
- statuses, 155

### Std Deviation
- 112

### Step
- 33
- step, 36
- sTitle, 76, 77, 107

### sToolName
- 80, 82, 139

### stop
- 129

### stopRT
- 136

### SubmitLogin
- 57

### Subtract
- 111

### sUserName
- 85

### Tabbed Dialog Control object
- 140–142

### tableName
- 49

### tableNameMask
- 49

### tables
- 41, 49

### tag
- 34, 38

### Tag Search
- 79, 84

### Tag Search Dialog
- 88

### tag#
- 31, 32

### tag##
- 53

### tag0
- 34, 39

### tag1
- 39

### tagAttributes
- 52

### tagname
- 154

### tagNameMask
- 39

### tagnames
- 154

### tagSearch
- 52

### timeBounds
- 34

### timeZone
- 34, 36

### timezone
- 58, 60, 62, 63

### Title
- 104

### Toolbar object
- 136–138

### toolbarconfig.xml file
- 169

### Training Exercises
- 193

### Trend Control Bar object
- 134–136

### trend events
- 120

### OnAxisClick
- 125

### OnClick
- 120

### OnCursor
- 126

### OnDataLoaded
- 124

### OnDeleteCursor
- 126

### OnDoubleClick
- 121, 122

### OnError
- 122

### OnLegendClick
- 124

### OnLegendMouseMove
- 125

### OnTrendLoaded
- 124

### OnUnZoom
- 124

### OnZoom
- 123

### TrendDoubleClick
- 122

### trTrend
- 120, 123, 125, 126

### Tutorial
- 187

### Building an Application
- 187
U
undo, 136
unselectAllRow, 145
update, 135
URL queries, 27
  format specifiers, 27
  syntax, 27
useIL, 58
username, 58, 60
V
value, 154
values, 155
variable0, 54
variable99, 54
VBScript, 23
VersionInfo, 61
W
WdsData, 161
WebDataSet, 95
  applet element, 95–102
  applet parameters, 95
WebDataSet objects
  JWebDataSet object, 96
  JWebRecordSet object, 98, 151
  XField object, 98
WriteStringBlob, 152, 153
WriteTag, 153
WriteTags, 153, 154
WriteVariantBlob, 152, 153
writing BLOB data using NetDAC, 152–153
writing tag values using NetDAC, 153–155
X
X, 108
XField object, 98
XML, trend, 104
X-Y Plot, 112
Y
Y, 108