Autodesk® Topobase™

Autodesk® Topobase™ Developer's Guide

Autodesk®

© **2009 Autodesk, Inc. All Rights Reserved.** Except as otherwise permitted by Autodesk, Inc., this publication, or parts thereof, may not be reproduced in any form, by any method, for any purpose.

Certain materials included in this publication are reprinted with the permission of the copyright holder.

Trademarks
The following are registered trademarks or trademarks of Autodesk, Inc., in the USA and other countries: 3DEC (design/logo), 3December, 3December.com, 3ds Max, ADI, Alias, Alias (swirl design/logo), AliasStudio, Alias|Wavefront (design/logo), ATC, AUGI, AutoCAD, AutoCAD Learning Assistance, AutoCAD LT, AutoCAD Simulator, AutoCAD SQL Extension, AutoCAD SQL Interface, Autodesk, Autodesk Envision, Autodesk Insight, Autodesk Intent, Autodesk Inventor, Autodesk Map, Autodesk MapGuide, Autodesk Streamline, AutoLISP, AutoSnap, AutoSketch, AutoTrack, Backdraft, Built with ObjectARX (logo), Burn, Buzzsaw, CAiCE, Can You Imagine, Character Studio, Cinestream, Civil 3D, Cleaner, Cleaner Central, ClearScale, Colour Warper, Combustion, Communication Specification, Constructware, Content Explorer, Create>what's>Next> (design/logo), Dancing Baby (image), DesignCenter, Design Doctor, Designer's Toolkit, DesignKids, DesignProf, DesignServer, DesignStudio, Design|Studio (design/logo), Design Web Format, Discreet, DWF, DWG, DWG (logo), DWG Extreme, DWG TrueConvert, DWG TrueView, DXF, Ecotect, Exposure, Extending the Design Team, Face Robot, FBX, Filmbox, Fire, Flame, Flint, FMDesktop, Freewheel, Frost, GDX Driver, Gmax, Green Building Studio, Heads-up Design, Heidi, HumanIK, IDEA Server, i-drop, ImageModeler, iMOUT, Incinerator, Inferno, Inventor, Inventor LT, Kaydara, Kaydara (design/logo), Kynapse, Kynogon, LandXplorer, LocationLogic, Lustre, Matchmover, Maya, Mechanical Desktop, Moonbox, MotionBuilder, Movimento, Mudbox, NavisWorks, ObjectARX, ObjectDBX, Open Reality, Opticore, Opticore Opus, PolarSnap, PortfolioWall, Powered with Autodesk Technology, Productstream, ProjectPoint, ProMaterials, RasterDWG, Reactor, RealDWG, Real-time Roto, REALVIZ, Recognize, Render Queue, Retimer,Reveal, Revit, Showcase, ShowMotion, SketchBook, Smoke, Softimage, Softimage|XSI (design/logo), SteeringWheels, Stitcher, Stone, StudioTools, Topobase, Toxik, TrustedDWG, ViewCube, Visual, Visual Construction, Visual Drainage, Visual Landscape, Visual Survey, Visual Toolbox, Visual LISP, Voice Reality, Volo, Vtour, Wire, Wiretap, WiretapCentral, XSI, and XSI (design/logo).

The following are registered trademarks or trademarks of Autodesk Canada Co. in the USA and/or Canada and other countries: Backburner,Multi-Master Editing, River, and Sparks.

The following are registered trademarks or trademarks of MoldflowCorp. in the USA and/or other countries: Moldflow, MPA, MPA (design/logo),Moldflow Plastics Advisers, MPI, MPI (design/logo), Moldflow Plastics Insight,MPX, MPX (design/logo), Moldflow Plastics Xpert.

All other brand names, product names or trademarks belong to their respective holders.

Disclaimer
THIS PUBLICATION AND THE INFORMATION CONTAINED HEREIN IS MADE AVAILABLE BY AUTODESK, INC. "AS IS." AUTODESK, INC. DISCLAIMS ALL WARRANTIES, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE REGARDING THESE MATERIALS.

Published by:
Autodesk, Inc.
111 Mclnnis Parkway
San Rafael, CA 94903, USA

Contents

Chapter 1 Introduction 1

What This Guide Covers 1
What is Topobase? 1
Preparing to Run the Examples 2
Creating Your First Plugin 2
Autodesk Topobase and Visual Studio 3
Installing Project and Item Templates 3
Installing User Controls 5

Chapter 2 New and Changed API Content in Topobase 2010 7

Wizard framework 7
Security Administrator API 9
Feature Class Forms Designer Class 10
New Feature Templates API 11
Changes in the Topobase.Utilities Namespace 13
New Split and Join Methods in Class Topobase.Data.FeatureClass 15
Import and Export Plot and Plot Templates 17
Point Numbering 19
API Samples 20
Obsolete or Deprecated APIs 21

Chapter 3 API Overview 23

Database . . . 23
Connecting to the Database . . . 23
Creating a New Database User . . . 24
Feature Classes . . . 26
About Feature Classes . . . 26
Topics . . . 26
Creating Feature Classes . . . 27
Feature Class Types . . . 28
Attributes . . . 29
Events . . . 30
Feature Class Sample . . . 31
Features . . . 33
About Features . . . 33
Creating Features . . . 33
Geometry . . . 34
Graphic Namespace . . . 34
Geometries . . . 35
Accessing a Feature's Geometry Property . . . 36
PlaneGeometry Namespace . . . 38
Overview of PlaneGeometry . . . 39
Creating PlaneGeometry Objects From Graphic Objects . . . 40
Moving, Scaling, and Rotating PlaneGeometry Objects . . . 41
SpaceGeometry Namespace . . . 43
Overview of SpaceGeometry . . . 43
Creating SpaceGeometry Objects From Graphic Objects . . . 44
Creating Graphic Objects From Geometry . . . 45
Topologies . . . 46
Topology Related Assemblies . . . 46
Logical Topologies . . . 47
LogicalTopology as Namespace and Class . . . 47
Initializing Logical Topologies . . . 48
Area Topologies . . . 48
Initializing Area Topologies . . . 49

Chapter 4 Quick Guide: Creating a Topobase Plugin . . . 51

Introduction . . . 51
Creating a New Project . . . 52
Accessing the Project Properties . . . 53
Configuring the Project to Write the Build Outputs to the Topobase Bin Folder . . . 54
Configuring the Project to Run Topobase from Visual Studio . . . 55
Adding References To Your Project . . . 56
Inspecting the Assembly Information . . . 57

Adding Code To The Default Class 57
Creating a Topobase Plugin Definition File 59
Testing the Plugin 60

Chapter 5 Creating Application Modules 63

Creating the Data Model 63
About Data Models 63
Creating the Structure Update Plugin 64
Creating an Update Version Module 67
Adding Changes to the Data Model 70
Data Model Elements 73
Topics 73
Feature Classes 73
Labels 74
Attributes 74
Attribute Relations 75
Domains 76
Utility Model 77
Client-Side Feature Rules 77
Workflows 78
Installing an Application With a Structure Update Plugin 79
Creating a Stand-Alone Topobase Extension 79
Adding Feature Rules 80
About Feature Rules 80
Creating Feature Rules 80
Creating the FeatureRulePlugin Module 80
Creating the Rules 81
Rule Priorities 82
Installation 83
Creating Workflows 84
About Workflows 84
Workflow Scripts in Topobase Administrator 84
Use of the “Me” Object 85
Workflow Plugin 86
Create a Workflow Plugin Module 86
Create a Workflow Pane UI 89
Acquisition Workflows - Basics of Digitizing 100
Analysis Workflows - Basics of Analysis 102
Installation 103
Creating User Interface Plugins 105
Creating an Application Plugin 106
About Application Plugins 106
Responding to Application Events 107
Adding Custom Toolbars and Toolbar Buttons 107
Adding Menu Items 109
Creating a Document Plugin 110

About Document Plugins . . . 110
Creating Custom Toolbars and Toolbar Icons . . . 111
Modifying the Context Menu . . . 112
Creating a Dialog Plugin . . . 114
About Dialog Plugins . . . 114
Creating Toolbar Items . . . 115
Creating a Menu Bar . . . 115
Adding Controls to a Dialog . . . 116
Creating a Application Flyin . . . 117
About Application Flyins . . . 117
Creating a Document Flyin . . . 119
About Document Flyins . . . 119
Creating a Dialog Flyin . . . 120
About Dialog Flyins . . . 120
User Interaction . . . 122
Using Forms . . . 122
Topobase Forms Controls . . . 122
Progress Bar . . . 124
Message Box and Input Box . . . 126
Installation . . . 128
Plugins and Visual Studio . . . 129
Installing the Project Template . . . 129
Installing the Item Templates . . . 130
Installing User Controls . . . 130
Creating Survey File Format Plugins . . . 131
Creating a Survey File Format Module . . . 131
Implementing StartImport . . . 132
Creating Option Pages . . . 135
Creating an Option Page Module . . . 135
Reading and Writing User Settings . . . 135
Installation . . . 138
Creating Administrator Pages . . . 139
Creating an Administrator Page Plugin . . . 139
Responding to Administrator Events . . . 140
Installing Administrator Plugins . . . 140
Installation . . . 141
Building and Installing an Application Module . . . 141
Creating a Document Using the Module . . . 141
Manually Installing a Stand-Alone Workflow . . . 144
Installing a Stand-Alone Plugin . . . 145

Chapter 6 Developer Samples . . . 147

Introduction . . . 147
Building The Samples . . . 148
Usernames and Passwords . . . 148
Topobase Database Server (TBSYS) . . . 148

Running The Samples 148
Common Steps 149
Details 149
Sample 01 - Create Structure 149
Purpose 149
Procedure 150
Create a Workspace 152
Create a Drawing Template 155
Sample 02 - Read/Write Features 159
Purpose 159
Procedure 159
Sample 03 - Oracle Data Provider (ODP) .Net 161
Purpose 161
Procedure 161
Sample 10 - Document Context Menu 163
Purpose 163
Procedure 163
Sample 12 - Document Toolbar Button 164
Purpose 164
Procedure 164
Sample 13 - Dialog Toolbar Button 166
Purpose 166
Procedure 166
Sample 14 - Main Menu 167
Purpose 167
Procedure 167
Sample 15 - Main Toolbar 169
Purpose 169
Procedure 169
Sample 16 - Dialog Button 171
Purpose 171
Procedure 171
Sample 17 - Dialog Menu 174
Purpose 174
Procedure 174
Sample 18 - Dialog Events 175
Purpose 175
Procedure 175
Sample 19 - Input Box, Confirm Box, and Multiple Input Box 176
Purpose 176
Procedure 177
Sample 20 - Forms and Controls 178
Purpose 178
Procedure 178
Sample 22 - Document Flyin 179

Purpose 179
Procedure 179
Sample 25 - Progress Bar 181
Purpose 181
Procedure 181
Sample 28 - List Box With Context Menu 181
Purpose 181
Procedure 181
Sample 29 - Tree View 182
Purpose 182
Procedure 182
Sample 30 - Map Button 183
Purpose 183
Procedure 183
Sample 31 - List Box 185
Purpose 185
Procedure 185
Sample 32 - Highlight 186
Purpose 186
Procedure 186
Sample 33 - Information 187
Purpose 187
Procedure 187
Sample 34 - File Upload and Download 188
Purpose 188
Procedure 189
Sample 37 - Windows Menu Items 189
Purpose 189
Procedure 190
Sample 40 - Create All Dialogs Boxes 192
Purpose 192
Procedure 192
Sample 41 - Grid Control 193
Purpose 193
Procedure 193
Sample 43 - Dialog Box Validation 194
Purpose 194
Procedure 194
Sample 44 - Menus and Toolbars 196
Purpose 196
Procedure 196
Sample 45 - Matrix 196
Purpose 196
Procedure 197
Sample 46 - Interaction 197
Purpose 197

Procedure 198
Sample 47 - Application Options 199
Purpose 199
Procedure 200
Sample 48 - Treeview With Context Menu 201
Purpose 201
Procedure 201
Sample 50 - Date/Time Picker 202
Purpose 202
Procedure 202
Sample 51 - File and Directory Text Box 203
Purpose 203
Procedure 203
Sample 52 - Dock In Container 204
Purpose 204
Procedure 204
Sample 53 - Connection Tools 207
Purpose 207
Procedure 208
Sample 55 - Drop Down Buttons 210
Purpose 210
Procedure 211
Sample 56 - List Box 211
Purpose 211
Procedure 211
Sample 57 - Special Menu Items 212
Purpose 212
Procedure 212
Sample 59 - Settings and Document Options 213
Purpose 213
Procedure 213
Sample 61 - Dialog Box User Controls 214
Purpose 214
Procedure 215
Sample 62 - Control Test 218
Purpose 218
Procedure 218
Sample 64 - Desktop Option Pages 219
Purpose 219
Procedure 219
Sample 66 - Reports 221
Purpose 221
Procedure 221
Sample 67 - Dialog Box FlyIn 223
Purpose 223
Procedure 223

Sample 68 - Option Page With Tree Nodes 224
Purpose 224
Procedure 225
Sample 69 - FlyIn Container 226
Purpose 226
Procedure 226
Sample 70 - Application Flyin 229
Purpose 229
Procedure 229
Sample 71 - Checked List Box 230
Purpose 230
Procedure 230
Sample 72 - Application Toolbar 231
Purpose 231
Procedure 231
Sample 74 - List Box Multi Selection 232
Purpose 232
Procedure 232
Sample 75- SQL Export 233
Purpose 233
Procedure 233
Sample 77 - Menu Control 234
Purpose 234
Procedure 234
Sample 78 - Web Browser 235
Purpose 235
Procedure 235
Sample 79 - Tab Control 236
Purpose 236
Procedure 236
Sample 80 - Canvas Control 237
Purpose 237
Procedure 237
Sample 81 - Dialog Box Control Update 238
Purpose 238
Procedure 238
Sample 83 - Remote Control 240
Purpose 240
Procedure 240
Sample 85 - Workflows 242
Purpose 242
Procedure 242
Sample 86 - Update Plugin 244
Purpose 244
Procedure 244
Sample 87 - Jobs 249

Purpose 249
Procedure 249
Sample 88 - Features 254
Purpose 254
Procedure 255
Sample 100 - Map Functions 257
Purpose 257
Procedure 257
Sample 102 - Picture Combobox 259
Purpose 259
Procedure 259
Sample 103 - Call Class Via Batch File 260
Purpose 260
Procedure 260
Sample 104 - Script Engine 262
Purpose 262
Procedure 262
Sample 105 - Create Topobase User 263
Purpose 263
Procedure 263
Sample 106 - Unit Support 264
Purpose 264
Procedure 264
Sample 111 - Feature Explorer 266
Purpose 266
Procedure 266
Sample 112 - Client Side Feature Rules 268
Purpose 268
Procedure 268
Sample 116 - Advanced Workflows 270
Purpose 270
Installing the Workflow 270
Procedure 271
Sample 117 - Simple Water Module 272
Purpose 272
Procedure 273
Sample 118 - Electric Templates 276
Purpose 276
Procedure 276
Sample 119 - Survey Import Plugin 280
Purpose 280
Procedure 280
Sample 120 - Custom URL Invoking 281
Purpose 281
Procedure 281
Sample 121 - Web Service 281

Purpose . . . 281
Procedure . . . 282
Sample 122 - Custom Point Numbering . . . 282
Purpose . . . 282
Procedure . . . 283
Sample 123 - Wizard Using WPF . . . 284
Purpose . . . 284
Procedure . . . 284

Appendix A TBP File Format . . . 285
Introduction . . . 285
Example TBP Files . . . 285
Plugin Tag Details . . . 287
Property Details . . . 288

Appendix B Object Models . . . 291
Legend . . . 291
Framework . . . 292
Data Namespace . . . 292
Data.Explorer Namespace . . . 294
Data.Jobs3 Namespace . . . 295
Data.Provider Namespace . . . 296
Data.Reference and Data.Util.OperationLog Namespaces . . . 297
Data.Sys Namespace . . . 298
Graphic Namespace . . . 299
Math Namespace . . . 300
PlaneGeometry Namespace . . . 301
SpaceGeometry Namespace . . . 302
Data.Doc.Topologies Namespace . . . 303
Workflows . . . 304
Workflows Namespace . . . 304
Area Topology . . . 305
AreaTopology Namespace . . . 305
LogicalTopology Namespace . . . 306
Vertical Application Modules . . . 308
Common Namespace . . . 308
Electric.Common Namespace . . . 309

Index . . . 311

Introduction 1

What This Guide Covers

This guide describes how to use the Autodesk® Topobase™ API and associated sample code.

It assumes you have read applicable ReadMe files, reviewed appropriate application Help (as needed), and are familiar with using Autodesk® Topobase Client or Autodesk Topobase Web. All examples also assume that you have installed the sample data and sample applications supplied with Autodesk Topobase Client, which are included with the default installation package.

For more detailed information about the individual APIs, see the appropriate *Autodesk Topobase API References* located in the *<topobase>\Help* directory:

- Topobase_Client_API.chm
- Topobase_Modules_API.chm
- Topobase_ServerSide_API.chm

For more information about administration tasks for Autodesk Topobase, such as creating and editing data models, setting up users and application options, or customizing feature class forms, see *Autodesk Topobase Administrator Guide*.

What is Topobase?

Autodesk Topobase is an infrastructure asset management solution that provides centralized, flexible, and secure access to spatial information for planning, design, operations and business teams. Built on AutoCAD Map 3D, Autodesk MapGuide Enterprise, and Oracle software, Autodesk Topobase helps you see

the big picture and make better decisions by integrating CAD, map, asset, GIS and customer information for a more comprehensive view of your infrastructure.

Topobase Plugins

Plugins play an important role in Autodesk Topobase. They allow you to extend and customize Topobase to fit the needs of your organization. For example, they enable customization of the user interface, create and update data models, provide management of workspaces, and control how users interact with documents and workspaces.

In fact, many of the Autodesk Topobase components are plugins created by Autodesk developers. It enables developers to extend Topobase for their own needs. To create your own plugin, you need to create a *.dll* with one or more plugin classes in it and create a *.tbp* file which lists the plugin classes available in the library. See TBP File Format (page 285) for more information.

Preparing to Run the Examples

Autodesk Topobase provides a large number of samples demonstrating many different kinds of plugins and showing how to perform simple tasks using the API. These are provided as Microsoft® Visual Studio® 2005 projects in both C# and Visual Basic formats.

The section Developer Samples (page 147) describes how to compile, install, and run the samples, and gives a brief description of what each sample does. The samples are numbered, but since obsolete samples have been removed the numbers are not always consecutive. Unless otherwise stated, all the samples can be run in both Autodesk Topobase Client and Autodesk Topobase Web Client.

Creating Your First Plugin

Quick Guide: Creating a Topobase Plugin (page 51) describes how to code, configure, and run a simple C# or Visual Basic user interface plugin project. The process creates a Topobase plugin that generates the familiar “Hello World” output. This basic instruction provides a foundation for using and creating other plugins and for exercising and customizing the many sample applications provided.

Autodesk Topobase and Visual Studio

Installing Project and Item Templates

Autodesk Topobase includes a project template for creating stand-alone plugins in Visual Studio 2005. It also includes a series of item templates for adding new workflows, update plugins, user interface plugins, and other classes and forms to an application module project or stand-alone plugin project.

Installing the Plugin Project Template

Autodesk Topobase includes a generic project template for creating stand-alone plugin projects in Visual Studio 2005. To install it, copy the project template named *CSSimpleTopobasePlugIn.zip* from the *<Topobase install directory>\Development\VS Templates\ProjectTemplates\Topobase* directory to the Visual Studio custom template directory (usually *My Documents\Visual Studio 2005\Templates\ProjectTemplates\C#*).

To create a stand-alone plugin project using the template:

1. Start Visual Studio.
2. Select the File ➤ New ➤ Project menu item.
3. In the New Project dialog box, select the Visual C# Project Type.
4. Select SimpleTopobasePlugIn from the list of templates and create a project.
5. Once the project has been created, check the project references to make sure that the references to the Topobase libraries are valid.

Installing the Item Templates

Item templates provide a quick and standardized means to create Autodesk Topobase specific classes and forms.

Plugins can use regular Windows forms and user controls to interact with the user. However, any plugin that uses these will only work in the desktop environment and will not work in the Autodesk Topobase web client. To create user interface elements that are usable in both environments, you can use form templates provided by Autodesk Topobase in your projects. These forms are limited to the controls provided in the *Topobase.Form.dll* library.

To install the templates, copy the item template files (which are all *.zip* compressed files) from the *<Topobase install directory>\Development\VS Templates\ItemTemplates\Topobase* directory to the appropriate Visual Studio custom template directory (usually *My Documents\Visual Studio 2005\Templates\ItemTemplates\Visual C#* and *My Documents\Visual Studio 2005\Templates\ItemTemplates\Visual Basic*).

To add Topobase forms to your project:

1. Select the Project ➤ Add New Item menu item.
2. Select the appropriate plugin component type from the available templates in the My Templates section of the template list.

The following is a list of the types of items available:

- **DocumentPlugIn** - Creates an class of type `Topobase.Forms.DocumentPlugIn`. This serves as the base for user interface plugins which deal with individual documents. For more information, see Creating a Document Plugin (page 110).
- **StructureUpdatePlugin** - Creates an class derived from the abstract class `Topobase.Update.DocumentStructureUpdatePlugIn`. The structure update plugin provides an interface to the data model definitions and provides a mechanism for updating the data model when the data model is modified. For more information, see Creating the Structure Update Plugin (page 64).
- **StructureUpdateVersion** - Creates an class derived from the abstract class `Topobase.Update.StructureUpdateVersionBase`. The data model is defined in a series of update version classes. Each time you change the data model you will add a new StructureUpdateVersionBase-derived class which codifies the changes between the previous version and the latest version. For more information, see Creating a Update Version Module (page 67).
- **TopobaseApplicationForm** - Creates a standard form with title bar based on the `Topobase.Forms.ApplicationForm` class. These forms are used to create dialog boxes for user interface plugins.
- **TopobaseApplicationOptionPage** - Creates a user control based on the `Topobase.Forms.OptionPages.ApplicationOptionPage` class. These are used to add new pages to the Topobase Application Options dialog box. For more information, see Creating Option Pages (page 135).
- **TopobaseDocumentFlyIn** - Creates a user control based on the `Topobase.Forms.FlyIns.DocumentFlyIn` class. These serve as the container

for the controls in a document flyin. For more information, see Creating a Document Flyin (page 119).

- **TopobaseDocumentForm** - Creates a standard form with title bar based on the `Topobase.Forms.DocumentForm` class. These forms are used to create dialog boxes for user interface plugins.
- **TopobaseDocumentOptionPage** - Creates a user control based on the `Topobase.Forms.OptionPages.DocumentOptionPage` class. These are used to add new pages to the Topobase Document Options dialog box. For more information, see Creating Option Pages (page 135).
- **TopobaseUserControl** - Creates a user control based on the `Topobase.Forms.UserControl` class. These user controls are used for embedding controls within a Topobase user interface element. For example, a workflow might use a user control within the Topobase task pane..

Installing User Controls

Autodesk Topobase includes a number of controls for use in Autodesk Topobase forms, some of which replicate standard Windows user control and some of which are unique to Autodesk Topobase. To add Autodesk Topobase controls to the Visual Studio Toolbox:

1. Right-click in the Toolbox, and select the Choose Items... menu item.
2. Select Browse in the Chose Toolbox Items dialog box.
3. Select *Topobase.Forms.dll* from the Autodesk Topobase Client */bin/* directory.

New and Changed API Content in Topobase 2010

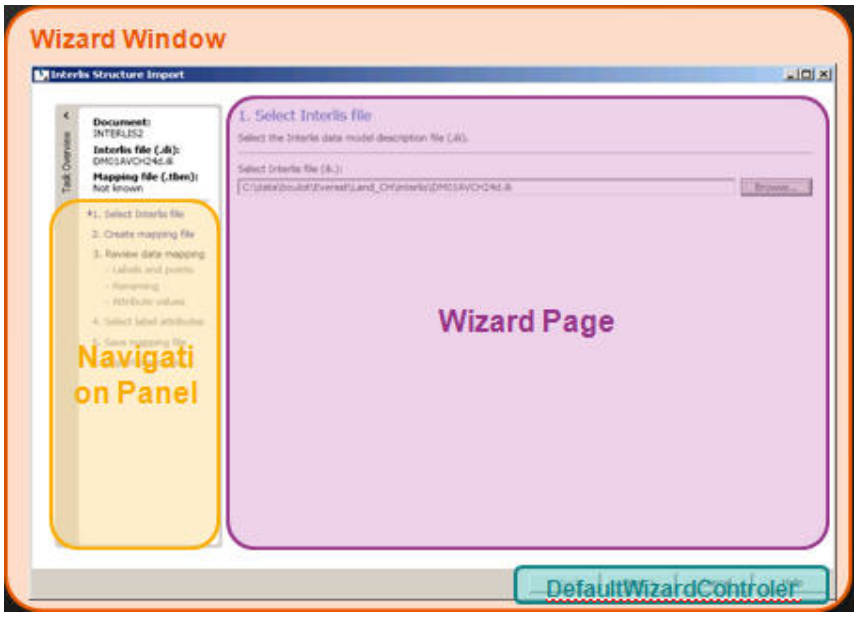
2

This topic provides a brief overview of some of the important changes to the Topobase 2009 application programming interface (API).

- Wizard framework
- Security Administrator API
- Design different forms per feature class
- New Feature Templates API
- Changes in the Topobase.Utilities namespace
- New split and join methods in class Topobase.Data.FeatureClass
- Import and export plot and plot templates
- Point numbering
- API samples
- Obsolete or deprecated APIs

Wizard framework

The subnamespaces of the Topobase.Forms.WpfControls namespace contains classes that comprize a new wizard framework in Topobase 2010. You can use the framework to create wizards for your Topobase applications that are consistent in operation and in look-and-feel.



Framework Components

The Wizard Window is the container for all of the framework's components:

- Navigation panel - The wizard framework builds this panel and fills it with the task overview — the steps for users to follow.
- Default wizard controller - This is another framework component. It provides the Back, Next, Finish, and Cancel buttons. You can overwrite the wizard controller to customize its capabilities.
- Wizard pages - WIzard pages define the wizard's user interface. Wizard pages can be base pages, on which users perform actions, or automatic wizard pages, where the wizard framework performs actions.

Sample Code

To take adavantage of the wizard framework, you need only create and organize a navigation panel and implement a WizardDataObject, the wizard framework provides all of the remaining capabilities. The <Topobase-installation-directory>/Development/Samples contains an API sample (sample number 123) that demonstrates how to use the wizard framework.

The following code snippet shows the basic steps required to use the wizard framework to create a new Topobase Wizard.

```
// Instantiate a NavigationPanel object to contain and organize
// the steps and substeps of the task you want the wizard to per
form.
// (Topobase.Forms.WpfControls.Navigation namespace)
NavigationPanel navigationPanel = new NavigationPanel();
// The WizardWindow object is the container for everything in
// the wizard.
// (Topobase.Forms.WpfControls.Wizard namespace)
WizardWindow window =
   new WizardWindow(Resources.Title, navigationPanel);
// The WizardDataObject provides access to a Topobase document
// and holds the logic needed to access wizard pages.
// (Topobase.Forms.WpfControls.Wizard namespace)
WizardDataObject dataObject =
   new WizardDataObject(Document);
// The DefaultWizardController manages public events in the
// wizard pages such as Back, Next, Finish, and Cancel.
// You can overwrite this bahavior.
// (Topobase.Forms.WpfControls.Wizard namespace)
DefaultWizardController wizardController =
   new DefaultWizardController(window.WizardControl, dataObject);
window.Show();
```

Security Administrator API

The Security Administrator is a new standalone tool in Topobase 2010. The tool provides advanced administrators with the capability to edit database permissions. Using the Security Administrator, you can configure a Topobase document, by workspace and user group, to customize user permissions.

You can work programatically to create roles and manage user privilieges using the *Topobase.Security.dll* assembly, which contains the API exposed in the `Topobase.Security` namespace.

Sample code

The following code snippet shows how to use the `Topobase.Security.Security` class to create a connection to the database, create a role and link it to a data user and Oracle guest user, and how to set and update permissions.

```
// Create a Topobase.Security.Security instance to manage user
// privileges.
// The Security class constructor creates a connection
// (Topobase.Data.Provider.Connection) to a database user.
_Security = new Security(connection, logfileName);

// Create a Topobase.Security.Role to link the data user and
// Oracle guest user.
Role role = _Security.Roles.Create("TestRoleName");

// Create a Topobase.Security.DataUser.  The DataUser can be a
// Topobase user.
// You can define templates in the Security Administrator tool
// or use one of the two predeined templates.
DataUser dataUser =
   role.DataUsers.Add("DataUserName", templateFile);

//Create a Topobase.Security.DbObject and set privileges.
DbObject priv = dataUser.DbObjects["MyTableAAA"];
priv.Privileges.Add(PrivilegeType.Select);
priv = dataUser.DbObjects["MyTableBBB"];
priv.Privileges.Add(PrivilegeType.Select);
priv.Privileges.Add(PrivilegeType.Update);

// Update all priviliges of the data user associated with the role.
role.UpdatePrivileges();

//Create a guest user.
role.GuestUsers.Create(
   "guestUserName", "avs", false, "USERS", "TEMP");
```

The Security Administratior API reference documentation is in the Topobase.Security namespace section of the Client API reference.

Feature Class Forms Designer Class

The Topobase 2010 Forms Designer in Topobase Administrator gives you the capability to assign to different user groups forms used to customize feature classes. Using the Forms Designer you can hide or show text boxes on a form, create controls, check boxes, radio buttons, and combo boxes, and organize and arrange elements to create a form that suits your needs. The Topobase Adminstrator Guide and Client User Guide contain information about the Forms Designer and detailed instructions on its use.

To work programmatically with feature class forms, use the *Topobase.Forms.dll* assembly and the API exposed in the Topobase.Forms namespace. The following API components in the `Topobase.Forms` namespace are new in Topobase 2010:

- `public void AssignToUserGroup(Int64 dialogId, String userGroupId);`
  This method assigns a dialog to a usergroup. You can assign the same dialog to multiple usergroups, but a usergroup can be associated with ony one layout. That is, a usergroup is assigned to only one dialog or layout per table. If you assign another dialog to a usergroup, you will loose the pervious assignment.
- `public Dialog Duplicate(Int64 dialogId, String newDialogLayoutName);`
  This method enables you to create a copy of an existing dialog.
- `public Boolean Exists(String name);`
  This method returns true if a dialog box already exists.
- `public Boolean Exists(String dialogName, String dialogLayoutName);`
  This method returns true if a dialog layout already exists.
- `public Dialog Item(Int64 dialogId);`
  This method identifies and returns a dialog given the dialog's ID.
- `public Dialog Item(String dialogName, String dialogLayoutName);`
  This method returns a dialog given the dialog's name and type name. Use this method to manage dialogs rather than to open them.
- `public void Remove(Int64 dialogId);`
  Use this method to kill a dialog and to remove its layout and related settings from the database.

New Feature Templates API

A feature template is a collection of features. A specific example of a feature template is a cross section template used in the Electic NA and Electric CE vertical application modules. You can configure feature templates to include different arrangements and numbers of features, such as the conductors and ducts in a cross section.

The Template Manager dialog in Topobase 2010 uses the API in the new Topobase_Templates_ImportExport_XmlTemplateImportSettings.htm namespace. The API gives you the capability to import, export, and duplicate feature templates among Topobase documents programatically.

Import

The important class for importing and working with imported feature templates is `Topobase.Templates.ImportExport.XMLTemplateImporter`. XML import and export is designed to work with feature templates in Topobase documents that have the same structure. The importing, exporting, and duplicating operations do not create client system tables.

Import behavior is controlled by a `Topobase.Templates.ImportExport.XmlTemplateImportSettings` object, which contains properties for managing duplicate template names, existing domain values, and reference rows.

The Template Import/Export API makes use of another new API in the `Topobase.ImportExport` namespace.

```
XmlTemplateImporter templateImporter = new XmlTemplateImporter(
this.document.Connection, /* Connection to destination document
*/
importSettings /* Optional*/);

templateImporter.ReadIn(
  importStream /* a FileStream */);

importedTemplates = templateImporter.ImportTemplates();
```

Export

The important class for working with exported feature templates is `Topobase.Templates.ImportExport.XMLTemplateExporter`. The following list shows the information that this process exports:

- Complete feature template records:
  TB_TEMPLATE
  TB_TEMPLATE_EXPLORER
  TB_TEMPLATE_FEATURE
  TB_TEMPLATE_RELATION
- Associated feature class information from TB_DICTIONARY
- Associated information from TB_RELATION
- Associated reference records
- All domain values from any referenced Domain table
- Document meta data:

Coordinate system
Module cods
Document version
TBSYS version
Application version

```
XmlTemplateExporter templateExporter =
  new XmlTemplateExporter(this.document.Connection);
  /* The connection to the source document */

// Add the template to the list of templates to be exported
foreach (Template template in checkedTemplates)
    templateExporter.AddTemplate(template);

// Finally export the templates to a stream, for
// example, a FileStream.
templateExporter.Export(exportStream);
```

Duplicate

Duplicating a template will not duplicate the template instances.

```
// Duplicate template by template - for each template a
//'TemplateName (x)' will be created
foreach (Template checkedTemplate in checkedTemplates)
{
    TemplateDuplicator duplicator =
      new TemplateDuplicator(this.document.Connection);

    Template newTemplate =
      duplicator.DuplicateTemplate(checkedTemplate);
    // Add the new template to the internal list of duplicated
    // templates.
    duplicatedTemplates.Add(newTemplate);
}
```

Changes in the Topobase.Utilities Namespace

In Topobase 2010, the `Topobase.Utilites` namespace has a new subnamespace, `Topobase.Utilities.App`. This new namespace contains some of the API components that were formerly available in the Topobase.Utilites namespace.

If your applications use these APIs, ensure that you include a reference to the new namespace in your projects.

The following table shows the high-level changes between the `Topobase.Utilites` namespace in Topobase 2009 and the `Topobase.Utilites` and `Topobase.Utilities.App` namespaces in Topobase 2010. Refer to the Client API reference for full details.

TB2009 Topobase.Utilities namespace	Topobase2010 Topobase.Utilities namespace	Topobase 2010 Topobase Utilities.App namespace
FeatueExplorer namespace		FeatueExplorer namespace
FeatureRules namespace		FeatureRules namespace
Forms namespace		Forms namespace
PlugIns namespace		PlugIns namespace
Tracing namespace		Tracing namespace
ElementFinder class	ElementFinder class	
InvalidFeatureClassTypeException class	InvalidFeatureClassTypeException class	
Maintenance class		Maintenance class
SoftSplitUtilityLine class	SoftSplitUtilityLine class	
UtilitiesException class	UtilitiesException class	
UtilityFeature class	UtilityFeature class	
UtiltityFeatureClass class	UtiltityFeatureClass class	

TB2009 Topobase.Utilities namespace	Topobase2010 Topobase.Utilities namespace	Topobase 2010 Topobase Utilities.App namespace
UtiltityFeatureClassFactory class	UtiltityFeatureClassFactory class	
UtillityFeatureComparer class	UtillityFeatureComparer class	
UtilityFeatureIdEqualityComparer class	UtilityFeatureIdEqualityComparer class	
UtilityLine class	UtilityLine class	
UtilityPoint class	UtilityPoint class	
		UtilityFeature-ClassExtensions
		UtilityLineExtensions class
		UtilityPointExtensions class

New Split and Join Methods in Class Topobase.Data.FeatureClass

The Topobase 2010 API contains new methods in class `Topobase.Data.FeatureClass` to split and join features.

Rather than performing split and join operations on your data with language operations such as insert, delete, and update, you can use the new methods in the `Topobase.Data.FeatureClass` class. These methods ensure that either all or none of the operations are performed.

Client side feature rules can also recognize these new methods as special operations. The event arguments used in feature rules (`Topobase.Events.FeatureEventArgs`, `Topobase.Events.FeatureCancelEventArgs`, `Topobase.Events.FidEventArgs`, and `Topobase.Events.FidCancelEventArgs`) have the following properties in Topopbase 2010:

```
bool IsPartOfJoin { get; }
Topobase.Events.FeatureJoinCancelEventArgs JoinEventArgs { get; }
bool IsPartofSplit { get; }
Topobase.Events.FeatureSplitEventArgs SplitEventArgs { get; }
```

You might decide to configure feature rules to examine these properties and react differently when the properties are set. For example, you can configure a feature rule that splits lines automatically to ignore operations that are part of the split to prevent the feature rule from calling itself recursively. You might also configure the feature rule to ignore operations that are part of a join to prevent undoing the operation that the user has specified.

The lists in the following subsection show the new split and join methods. Refer to the Client API reference for details.

New Split Methods

- `SplitFeature(Topobase.Data.Feature, Topobase.Graphic.Point, out Topobase.Data.Feature, out Topobase.Data.Feature)`
- `SplitFeature(Topobase.Data.Feature, Topobase.Graphic.Geometry, Topobase.Graphic.Geometry, out Topobase.Data.Feature, out Topobase.Data.Feature)`
- `SplitFeature(Topobase.Data.Feature, ref Topobase.Data.Feature, ref Topobase.Data.Feature)`
- `SplitFeature(Topobase.Data.Feature, ref Topobase.Data.Feature, ref Topobase.Data.Feature, bool, bool)`
- `SplitFeature(Topobase.Data.Feature, ref Topobase.Data.FeatureList, bool, bool`

New Join Methods

- `JoinFeatures(Topobase.Data.Feature, Topobase.Data.Feature, bool, out Topobase.Data.Feature)`
- `JoinFeatures(Topobase.Data.Feature, Topobase.Data.Feature, Topobase.Graphic.Geometry, out Topobase.Data.Feature)`
- `JoinFeatures(Topobase.Data.Feature, Topobase.Data.Feature, ref Topobase.Data.Feature)`
- `JoinFeatures(Topobase.Data.Feature, Topobase.Data.Feature, ref Topobase.Data.Feature, bool, bool)`

- `JoinFeatures(System.Collections.Generic.ICollection<Topobase.Data.Feature>, ref Topobase.Data.Feature, bool, bool)`

Import and Export Plot and Plot Templates

The capability to import and export plots and plot templates is new in Topobase 2010. This new feature enables users to exchange one or more plots and plot templates between Topobase documents.

To import and export plots and plot templates programmatically, the important classes to work with are `XmlPlotLibraryImporter` and `XmlPlotLibraryExporter` in the new namespace `Topobase.Plot.Data.ImportExport`.

Sample Code Snippets

The following code snippets show the basic steps for importing and exporting XML plot and plot templates to and from Topobase documents that have the Topobase Plot extension enabled. Note that to import a plot or plot template with an `XmlPlotLibraryImporter` object, the XML content must have been exported using the `XmlPlotLibraryExporter` class. Note also the use of a conflict handler to manage template naming conflicts.

The compoonents of the `Topobase.Plot.Data.ImportExport` namespace are well documented; refer to the Client API reference for more details.

Import

```
// Create an importer instance for the current IPlotConnection
XmlPlotLibraryImporter importer = new
   XmlPlotLibraryImporter(plotConnection);

// Read in the content and close the stream.
importer.ReadIn(inputStream);
inputStream.Close();

// Optionally create a conflict handler. The API client can
// then specify a call back to be used in case it
// encounters a naming conflict.
ConflictHandler conflictHandler =
   new ConflictHandler(... /* delegate, see API */);
importer.ConflictHandler = conflictHandler;

// Attach a listener to the PlotBaseImporting event so that
// the import can be canceled/monitored, for example by a
// progress bar.
importer.PlotBaseImporting +=
   ... /* event - to be used to cancel the import etc. */;

ICollection<PlotLibraryItem> importedItems;
// Run the import -- evaluate [importCanceled] if necessary.
bool importCanceled = importer.ImportAll(
   destGroup, maintainStructure, out importedItems);
```

Export

```
// Create an exporter instance for the current IPlotConnection
XmlPlotLibraryExporter exporter = new XmlPlotLibraryExporter(plot
Connection);

// Add groups recursively
exporter.AddGroup(libraryGroup, true /* recursive */);

// Export a single plot and its template, if it has one.
exporter.AddPlot(plot, true /* add template, too*/);

// Write the XML content to an arbitrary output stream and close
// the stream.
exporter.Export(fileStream);
fileStream.Close();
```

Point Numbering

The Point Numbering feature in Topobase 2010 has been implemented primarily for the Swiss Land Management vertical application module. The feature is not specific to Swiss Land Management module and you can use it with other vertical applications.

The point numbering feature provides the capability to gernerate point numbers automatically for several point feature classes during data aquisition. This helps to improve the efficiency of the data aquisition process, to reduce errors, and to manage the long and complex point number field.

In Topobase Administrator 2010, a new point numbering node appears in the document tree of a workspace in the Adminstrator Explorer. The standard implementation includes four point numbering types: per Document, per Group, Grid 100m, and Raster Plan. These are the most frequently used point numbering types, but you can create your own custom types as necessary. A point number group is a concatenation of point feature classes to which the same point numbering increment is applied.

API Components

The following new API components related to the point numbering feature enable you to work programmatically with the property pane and group tree views in Topobase Administrator:

- `Topobase.BaseFeatureRules.PointNumbering.Administrator.PointNumberAdminPage` class
  Contains methods for working with the point numbering property pane.
- `Topobase.Forms.UserControls.PointNumbering.PointNumberGroupTreeView`
  A user control that contains methods for working with the point numbering group tree view.

Feature Rules

A new feature rule group in Topobase 2010 adds the capability to set the point number automatically to the system attribute TB_POINTNUMBER.

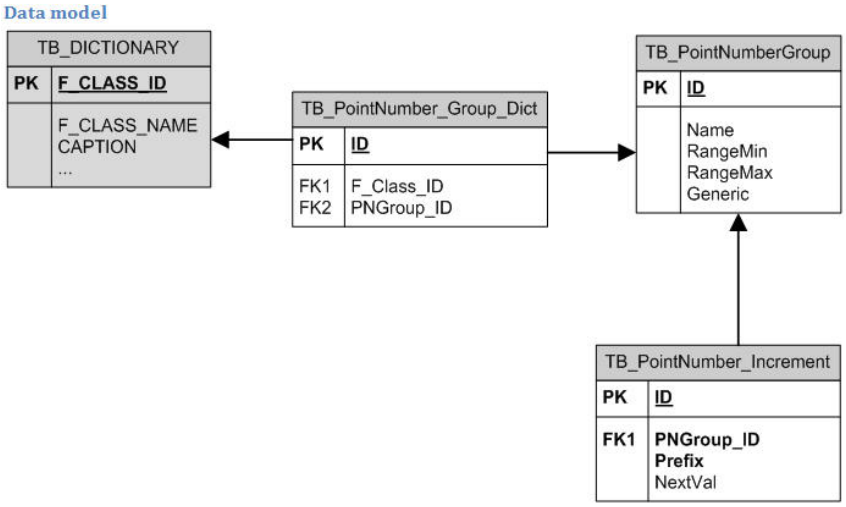
The rule group is called Automatic Point Number and contains the following feature rules:

- AutomaticPointNumber_BI
- AutomaticPointNumber_BU

These are client side feature rules that are applied before a feature is inserted or updated in the database. This rule group manages the complete point numbering mechanism and sets the new number or updates the increment sequence for given numbers.

Custom Point Number Types

Topobase 2010 includes three new system tables to record and store point numbering configuration and relationship information.



If you create your own custom point numbering type programatically using the API, do not access the system tables directly but use the `Topobase.Data.TBConnection.PointNumberGroups` property, which returns a `PointNumberGroupList`, a collection of point number groups. Refer to sample 122 in *<Topobase-installation-directory>/Development/Samples* which demonstrates this process.

API Samples

The *<Topobase-installation-directory>/Development/Samples* directory contains API samples in both C# and VB.NET. The samples have been revised for Topobase 2010 and new samples, 120, 121, 122, and 123, have been added.

Refer to the Developer's Guide (*<Topobase-installation-directory>/Help/TopobaseDevelopersGuide.chm*, for the HTML Help version or

<Topobase-installation-directory>/Manuals/TopobaseDevelopersGuide.pdf, for the PDF version) for detailed descriptions of all the API samples.

Obsolete or Deprecated APIs

If the source code for Topobase 2010 API components is decorated with .NET attributes, the API reference documentation shows the attribute information. This information identifies classes, methods, properties, and so on, that you should no longer use in your applications and refers you to preferred components. For example:

```
// .NET 'obsolete' attribute (C# syntax)
[Obsolete("Property is not supported anymore. Please use GetEnd
Point().", false)]
public UtilityPoint EndPoint;

// .NET 'obsolete' attribute (VB.NET syntax)
<Obsolete("Property is not supported anymore. Please use GetEnd
Point().", false)>
Public Property EndPoint() As UtilityPoint
```

API Overview

3

Database

This section covers basic database operations that are required for using the rest of the Topobase API.

Connecting to the Database

To connect to the database, initialize a new instance of the `TBConnection` class via its constructor and invoke its `Open()` method to open the connection. After successfully opening the connection, you can then proceed to carry out other tasks, such as creating feature classes and features. After completing all required tasks, close the connection by calling `TBConnection.Close()`. Note that the connection should also be disposed of when it is no longer used. The following code shows the procedure for connecting to Topobase:

```
using (Topobase.Data.TBConnection myConnection =
 new Topobase.Data.TBConnection(
 Properties.Settings.Default.Oracle_User,
 Properties.Settings.Default.Oracle_Password,
 this.Application.Connection.DataSource,
 true,
 this.Application.Connection))
{
 // Open the connection to the database.
 // If the DB has no database structure it will be created now
 myConnection.ShowForms = true;
 myConnection.Open();

 // Tell the user what the connection state is.
 string state = myConnection.State.ToString();
 MessageBox.Show("Connection state:" + state);

 // Close connection.
 myConnection.Close();
}
```

Creating a New Database User

To create a new database user, use the `TBConnection.CreateTopobaseUser()` method. This method takes as one of its parameters a `Connection` object that corresponds to the current user name and password.

This code snippet shows how to open a connection and create a database user:

```
// Create a new empty Oracle user.
string userName = Properties.Settings.Default.Oracle_User;
if (!this.Application.Connection.ExistsUser(userName))
{
 // The name of the new database and the password are stored
 // in the project settings.
 using (Topobase.Data.TBConnection myConnection =
 new Topobase.Data.TBConnection(
 Properties.Settings.Default.Oracle_User,
 Properties.Settings.Default.Oracle_Password,
 this.Application.Connection.DataSource,
 true,
 this.Application.Connection))
 {

 try
 {
 myConnection.CreateTopobaseUser(
 this.Application.Connection,
 "USERS",
 "TEMP",
 "USERS");
 }
 catch (Topobase.Exception.TBException ex)
 {
 if (ex.Caller == Topobase.Exception.TBExceptionCaller.TB3Server)
 {
 MessageBox.Show("Unable to create user:"
 + Environment.NewLine + ex.Message);
 }
 else
 {
 throw;
 }
 }
 finally
 {
 myConnection.Close();
 }
 }
}
else
{
```

```
 MessageBox.Show("User already exists.");
}
```

Feature Classes

About Feature Classes

A feature class represents a type of Topobase object, and corresponds to a single table within the database. For example, the concept of "parcel" is a single feature class. The list of all feature classes within a database is contained within the `TBConnection` object's `FeatureClasses` property. To get a reference to an existing feature class, you can specify the feature id or string name as the index to the collection:

```
// Get a reference to the feature class named "MyPoint".
Topobase.Data.FeatureClass MyPointFeatureClass;
MyPointFeatureClass = myConnection.FeatureClasses["MyPoint"];
```

Topics

Topics are used to organize feature classes. A topic is a collection of feature class tables that have some relationship meaningful to the user. For example, all feature classes related to pipes can be placed within a single topic named "Pipes". Topics do not imply any table relationships between the feature classes contained within them. Topics may have subtopics as well. The collection of all topics within a database is contained within the `TBConnection` object's `Topics` property.

Topics are created by using the collection's `Add()` method. To create subtopics, you can either use the overridden `Add()` method that takes the parent topic id as a parameter, or you can directly add topics to the parent topic's `ChildTopic` property.

```
Topobase.Data.Topic MyTopic;
MyTopic = myConnection.Topics.Add("MyTopic", "Caption");
// Add a subtopic.
Topobase.Data.Topic SubTopic;
SubTopic = MyTopic.ChildTopics.Add("SubTopic", "Caption");
```

Creating Feature Classes

To create a new feature class, use the `FeatureClass` constructor to create a new object and then call the feature class object's `Create()` method to add the feature class to the database.

```
// Add a point-style feature class to the topic MyTopic.

Topobase.Data.FeatureClass MyPointFeatureClass;
// Do no create if it already exists.
if (!myConnection.FeatureClasses.Contains("MyPoint"))
{
 // Create the feature class object.
 MyPointFeatureClass = new Topobase.Data.FeatureClass(
 myConnection,
 "MyPoint",
 "My point caption",
 Topobase.Data.FeatureClassType.Point);
 MyTopic);

 // Add the feature class to the database and add references
 // of the object to the specified topic and other locations.
 MyPointFeatureClass.Create();
}
```

You can hierarchically organize feature classes by making one feature class a child of another feature class. To create a child feature class, use one of the constructor overrides where you specify the parent feature class id as one of the parameters. The following code sample creates a new feature class which is a child of the previously created `MyPointFeatureClass` feature class:

```
Topobase.Data.FeatureClass ChildFeatureClass;

ChildFeatureClass = new Topobase.Data.FeatureClass(
 myConnection,
 "MyPoint",
 "My point caption",
 Topobase.Data.FeatureClassType.Point);
 MyTopic,
 MyPointFeatureClass.ID);

ChildFeatureClass.Create();
```

You can see all the children of a feature class by accessing the `FeatureClass.ChildFeatureClasses` property, and you can find the parent of a feature class by accessing the `FeatureClass.ParentFeatureClass` property.

Feature Class Types

Feature classes are based on pre-defined types. These types define the default properties and geometry of the feature class. Types are defined in the `Topobase.Data.FeatureClassType` enumeration. The possible values are:

Point	Represents a standalone point.
Centroid	Represents the geographic center point of the associated polygon feature class.
LineString	Represents a polyline - a connected series of line and arc segments.
Polygon	Represents a closed figure made of lines, arcs, and LineString elements.
Label	Represents a text string located relative to a standalone point location.
Collection	Represents a feature class containing a combination of Points, LineStrings, or Polygons.
Attribute	Represents a feature class without geometry.
CompoundPolygon	Represents multiple Polygon or LineString features, each with different attributes.
CompoundLineString	Represents multiple LineString features, each with different attributes.
Dimension	Represents a feature class which can store the Dimension Features. It is child from an Attribute feature class and parent from a Label feature class.

Attributes

Feature class tables have a fixed basic structure depending on the type, but you can add additional columns called “attributes”. A feature class can have any number of attributes. The collection of attributes for a feature class object are access through the `FeatureClass.Attributes` property.

The following code will check to see if a name attribute already exists in the `MyPointFeatureClass` feature class. If not, it will add a new string attribute which can contain up to 255 characters.

```
if (!MyPointFeatureClass.Attributes.Contains("Name"))
{
 MyPointFeatureClass.Attributes.Add(
 "Name",
 Topobase.Data.DataType.Varchar2,
 255);
}
```

Attribute types are based on Oracle types, and are specified in the `Topobase.Data.DataType` enumeration:

Member of Topobase.Data.DataType enumeration	Type
Array	Oracle ARRAY
BFile	Oracle BFILE
Blob	Oracle BLOB
Boolean	(Only valid in PL/SQL procedures)
Char	Oracle CHAR
Clob	Oracle CLOB
Date	Oracle DATE
Geometry	Oracle SDO_Geometry (not supported in ODP.NET)
Long	Oracle LONG
NChar	Oracle NCHAR

Member of Topobase.Data.DataType enumeration	Type
NClob	Oracle NCLOB
Number	Oracle NUMBER
NVarchar2	Oracle NVARCHAR2
Raw	Oracle RAW
RowID	A RowID is a base 64-encoded physical address (location) of the row on the disk.
Table	Represents all Oracle Tables Types. (OraDirect.NET specific)
TimeStamp	Oracle TIMESTAMP
Varchar2	Oracle VARCHAR2

Events

Users can add and modify individual features within the feature class (that is, individual rows within the table) using the Topobase Client user interface. Your application may need to know when these changes take place. Feature class objects have a wide variety of events to notify you when these actions are taking place or have taken place.

The following example will add three new events to the `myFeatureClass` feature class to trigger functions whenever features are in the process of being inserted, have been inserted, or was canceled during the process of being inserted.

```
// Add three new events to myFeatureClass.
myFeatureClass.Inserting +=
 new Topobase.Events.FeatureCancelEventHandler
 (myFeatureClass_Inserting);

myFeatureClass.Inserted +=
 new Topobase.Events.FeatureEventHandler
 (myFeatureClass_Inserted);

myFeatureClass.InsertingCanceled +=
 new Topobase.Events.FeatureEventHandler
 (myFeatureClass_InsertingCanceled);


// This event handler will be invoked before a
// feature is inserted in myFeatureClass
private void myFeatureClass_Inserting(object sender, Topo
base.Events.FeatureCancelEventArgs e)
{
}
// This event handler will be invoked whenever a feature is
// inserted in myFeatureClass.
private void myFeatureClass_Inserted(object sender, Topo
base.Events.FeatureEventArgs e)
{
}
// This event handler will be invoked whenever a feature insertion
// in myFeatureClass is canceled.
private void myFeatureClass_InsertingCanceled(object sender, Topo
base.Events.FeatureEventArgs e)
{
}
```

Feature Class Sample

This sample demonstrates a typical action you might do with feature classes. It first creates a connection to the database, gets a reference to an existing feature class, and then adds a series of attributes to it.

```
// Connect to a Topobase.
Topobase.Data.TBConnection myConnection =
 new Topobase.Data.TBConnection(
 Properties.Settings.Default.Oracle_User,
 Properties.Settings.Default.Oracle_Password,
 this.Application.Connection.DataSource,
 true,
 this.Application.Connection);

myConnection.ShowForms = true;
myConnection.Open();

// Get an existing feature class.
Topobase.Data.FeatureClass MyPointFeatureClass;
MyPointFeatureClass = myConnection.FeatureClasses["MyPoint"];

// Add some attributes to the feature class.
if (!MyPointFeatureClass.Attributes.Contains("Name"))
{
 MyPointFeatureClass.Attributes.Add(
 "Name",
 Topobase.Data.DataType.Varchar2,
 255);
}

if (!MyPointFeatureClass.Attributes.Contains("Active"))
{
 MyPointFeatureClass.Attributes.Add(
 "Active",
 Topobase.Data.DataType.Number,
 1);
}

if (!MyPointFeatureClass.Attributes.Contains("Created"))
{
 MyPointFeatureClass.Attributes.Add(
 "Created",
 Topobase.Data.DataType.Date);
}

if (!MyPointFeatureClass.Attributes.Contains("Modified"))
{
 MyPointFeatureClass.Attributes.Add(
```

```
 "Modified",
 Topobase.Data.DataType.Date);
}

if (!MyPointFeatureClass.Attributes.Contains("Coordinates"))
{
 MyPointFeatureClass.Attributes.Add(
 "Coordinates",
 Topobase.Data.DataType.Varchar2,
 255);
}

// Close connection.
myConnection.Close();
```

Features

About Features

A feature represents an object in the real world. A feature is based on a feature class, and corresponds to a single row within a feature class table.

One way to access existing features is to use the `FeatureClass.GetFeatures()` method. This method returns a `FeatureList` object (a linked list of `Feature` objects using the `IList` interface). Certain overrides of this method allow filters and SQL `Where` clauses to limit the number of features returned.

Another way to get a list of features is to prompt the user to select them after they have generated graphics within the Topobase Client. Calling the `Topobase.Forms.Document.Map.GetFeatures()` method will pause execution of your program while the user selects features using the Map interface. It returns a linked list of features, which may be of different kinds of feature classes.

Creating Features

Call the `FeatureClass.CreateFeature()` method to create a new blank feature. This reserves a unique key (FID) for the feature.

```
// Get the FeatureClass
Topobase.Data.FeatureClass myPointClass =
 myConnection.FeatureClasses["MYPOINT"];

// Create a new Feature
Topobase.Data.Feature myPointFeature =
  myPointClass.CreateFeature();
```

Once the feature instance is created, set its properties. For example, you can add geometry to the feature through the `Feature.Geometry` property, as shown here:

```
myPointFeature.Geometry = new Topobase.Graphic.Point(100, 100);
```

Use the `FeatureClass.InsertFeature()` method to insert the new feature into Topobase (that is, create a new record in the feature class table).

```
myPointClass.InsertFeature(ref myPointFeature);
```

Use the `FeatureClass.UpdateFeature()` method to update an existing feature in Topobase.

```
myPointFeature.Geometry = new Topobase.Graphic.Point(200, 200);
// Now that a property has been changed, the feature needs to
// be updated.
myPointClass.UpdateFeature(ref myPointFeature);
```

To delete an existing feature, use the `FeatureClass.DeleteFeature()` method. This method requires the identifier of the feature to be deleted.

```
bool success = myPointClass.DeleteFeature(myPointFeature.ID);
```

Geometry

This section explains the geometry of features and the namespaces that help in examining, computing, and manipulating geometric objects.

Graphic Namespace

The `Topobase.Graphic` namespace contains the most important geometry-related classes. The `Geometry` property of Feature objects contains an object based on one of the classes in this namespace.

Be sure to reference *Topobase.Graphic.dll* in any project that uses this namespace.

Geometries

All of the geometries in the `Topobase.Graphic` namespace are derived from the abstract base class `Topobase.Graphic.Geometry` and contains features common to all the geometries. You will frequently see properties and parameters of type `Geometry` in the rest of the Topobase API.

The following table lists the different geometric classes in the namespace:

Class	Topology
LinePoint	A 3D point, used as part of a LineString or Polygon
LineString	A collection of LinePoints which form a single connected series of line segments.
MultiLineString	A collection of seperate LineStrings.
MultiPoint	A collection of Points.
MultiPolygon	A collection of Polygons.
Point	A 3D point.
Polygon	A LineString where the last point is also connected to the first point, forming a closed figure.
Unknown	A collection of ordinates. Can be used to store geometries that are not supported by Oracle Spatial.

All classes have a bounding box property of type `Topobase.Graphic.BoundingBox` which describes the furthest limits of the points in the X, Y, and Z dimensions. For classes that only contain a single point, all of the bounding box coordinates are set equal to the coordinates of the point.

These classes have very few methods for geometry analysis, limited to finding the distance between points or determining whether a point is located within a bounding box. For more advanced analysis, such as determining intersections and distances between geometries, you will need to convert the geometries into objects from the PlaneGeometry (2D) or SpaceGeometry (3D) namespaces.

The following code is from a plugin or workflow, and uses the plugin or workflow's base object to have the user select two points. The distance between the two points is determined.

```
Topobase.Graphic.Point point1 =
  this.Document.Map.GetPoint("Select Point 1");
Topobase.Graphic.Point point2 =
  this.Document.Map.GetPoint("Select Point 2");
double distance = point1.Distance2D(point2);
```

Accessing a Feature's Geometry Property

Point Geometry

The `Topobase.Graphic.Point` class represents a single 3D point location. The coordinates are stored in parameters named `X`, `Y`, and `Z`, which can also be set in the class constructor. The `Z` coordinate has a default value of zero.

The following code assigns a point with the coordinates "1234.5, 678.9, 0.0" to a feature:

```
pointFeature.Geometry = new Topobase.Graphic.Point(1234.5, 678.9);
```

LineString and Polygon Geometry

`Topobase.Graphic.LineString` and `Topobase.Graphic.Polygon` both consist of a collection of `Topobase.Graphic.LinePoint` objects. LinePoints are added to the collection one at a time using the `Add()` method (which adds the point to the end of the list of points) or the `Insert()` method (which inserts the point at the specified index in the existing list of points). LinePoints can also be added as a group by passing any collection of LinePoint objects that supports the `IEnumerable` interface (such as a `List` or `ArrayList`) in the LineString or Polygon constructor or to the `AddRange()` method.

Like a Point, a LinePoint represents a single 3D point location and has `X`, `Y`, and `Z` parameters. It also has an `Interpretation` parameter which describes the nature of the lines formed by the points. By default, LinePoints represent endpoints of straight line segments. LinePoints can also form arcs with one LinePoint representing the starting point of the arc (with the `Interpretation` parameter set to `Topobase.Graphic.Interpretations.ArcStart`), the next LinePoint representing a midpoint which the arc passes through (with the `Interpretation` parameter set to `Topobase.Graphic.Interpretations.ArcMidpoint`), and the last LinePoint representing the end point of the arc (with the `Interpretation` parameter set to `Topobase.Graphic.Interpretations.ArcEnd`). If two arcs are adjacent to each other, then the LinePoint seperating the two arcs should be set to `ArcStart`.

The following code creates a Polygon consisting of three straight line segments and assigns it to the `Geometry` property of a feature:

```
// Create the Polygon geometry.
Topobase.Graphic.Polygon polygonGeometry =
   new Topobase.Graphic.Polygon();

polygonGeometry.Add(new Topobase.Graphic.LinePoint(15, 27));
polygonGeometry.Add(new Topobase.Graphic.LinePoint(11, 26));
polygonGeometry.Add(new Topobase.Graphic.LinePoint(20, 11));

// Add Geometry to the Feature
polygonFeature.Geometry = polygonGeometry;
```

The following code creates a LineString consisting of two arc line segments and assigns it to the `Geometry` property of a feature:

```
// Create the LineString geometry.
Topobase.Graphic.LineString lineGeometry =
   new Topobase.Graphic.LineString();

lineGeometry.Add(new Topobase.Graphic.LinePoint(5, 10,
   Topobase.Graphic.Interpretations.ArcBegin));
lineGeometry.Add(new Topobase.Graphic.LinePoint(10, 26,
   Topobase.Graphic.Interpretations.ArcMidpoint));
// The end of the first arc and the start of the second.
lineGeometry.Add(new Topobase.Graphic.LinePoint(20, 13,
   Topobase.Graphic.Interpretations.ArcBegin));
lineGeometry.Add(new Topobase.Graphic.LinePoint(27, 7,
   Topobase.Graphic.Interpretations.ArcMidpoint));
lineGeometry.Add(new Topobase.Graphic.LinePoint(31, 20,
   Topobase.Graphic.Interpretations.ArcEnd));

// Add Geometry to the Feature
lineFeature.Geometry = lineGeometry;
```

Polygons can also contain holes. Each hole is defined by a separate Polygon object. New holes are added to the collection of holes using the `AddHole()` method.

```
// Create a triangular polygon for the hole.
List<LinePoint> lpList = new List<LinePoint>();

lpList.Add(new LinePoint(5, 5));
lpList.Add(new LinePoint(15, 5));
lpList.Add(new LinePoint(10, 15));

Polygon holePolygon = new Polygon(lpList);

// Assume outerPolygon already created and defined with line
// segments that fully enclose holePolygon.
outerPolygon.AddHole(holePolygon);
```

MultiLineString Geometry

`Topobase.Graphic.MultiLineString` consists of a collection of `Topobase.Graphic.LineString` objects. LineStrings are added to the collection one at a time using the `Add()` method (which adds the LineString to the end of the list of LineStrings) or the `Insert()` method (which inserts the LineString at the specified index in the existing list of LineStrings). LineStrings can also be added as a group by passing any collection of LineString objects that supports the `IEnumerable` interface (such as a `List` or `ArrayList`) in the MultiLineString constructor.

MultiPolygon Geometry

`Topobase.Graphic.MultiPolygon` consists of a collection of `Topobase.Graphic.Polygon` objects. Polygons are added to the collection one at a time using the `Add()` method (which adds the polygon to the end of the list of polygons) or the `Insert()` method (which inserts the polygon at the specified index in the existing list of polygons). Polygons can also be added as a group by passing any collection of `Polygon` objects that supports the `IEnumerable` interface (such as a `List` or `ArrayList`) in the MultiPolygon constructor.

PlaneGeometry Namespace

The `Topobase.PlaneGeometry` namespace consists of a series of classes that represent simple 2D geometric objects. These classes allow you to perform many kinds of geometric analysis not possible with the objects in the `Topobase.Graphic` namespace.

Be sure to reference *Topobase.PlaneGeometry.dll* in any project that uses this namespace.

Overview of PlaneGeometry

`Topobase.PlaneGeometry` includes classes for many different kinds of simple and compound figures. These classes are all immutable, meaning that they cannot be added to or modified after they have been constructed with the exception of inner rings for polygons. Figures are created by either using the class constructor or by using the static `TryCreate()` method in each of the following classes.

Class Name	Description
ArcSegment2D	An arc connecting two points, which can form part of a Line2D or Polygon2D.
Circle2D	A circle, created using either a point and a radius, or an arc segment.
Line2D	A contiguous series of ArcSegment2D and StraightSegment2D objects.
LineSegment2D	Abstract base class of ArcSegment2D and StraightSegment2D.
PackedLine2D	A Line2D converted to take up less memory, but which does not allow you to directly access the constituent segments.
PackedPolygon2D	A Polygon2D converted to take up less memory, but which does not allow you to directly access the constituent segments.
Polygon2D	A continuous series of ArcSegment2D and StraightSegment2D objects forming a closed figure. Holes can be added after the polygon has been constructed.
Straight2D	A straight line.

Class Name	Description
StraightSegment2D	A straight line segment connecting two points, which can form part of a Line2D or Polygon2D.

There are also two structures representing the basic elements of position and direction.

Structure Name	Description
Point2D	A structure representing a point location.
Vector2D	A structure representing a vector.

Creating PlaneGeometry Objects From Graphic Objects

Some of the classes in the `Topobase.Graphic` namespace have methods that directly create equivalent PlaneGeometry objects that have been flattened on the Z axis.

Topobase.Grpahic object	Method for creating Topobase.PlaneGeometry object
Point	ToPoint2D()
LinePoint	ToPoint2D()
LineString	ToLine2D()
Polygon	ToLine2D(), ToPolygon2D()

These methods make it easy to perform analysis and mathematical operations on `Topobase.Graphic` objects. The following example determines the area of a `Topobase.Graphic.Polygon` object:

```
// Given a valid Topobase.Graphic.Polygon object named tmpPoly,
// compute the area of this polygon.

Topobase.PlaneGeometry.Polygon2D poly2D = tmpPoly.ToPolygon2D();
double area = poly2D.CalculateArea();
```

Those classes that are made up of collections of single geometries (MultiPoint, MultiLine, MultiPolygon) do not have direct methods for creating PlaneGeometry objects. Instead you will need to look through all the items in the collection and call the method for creating those objects for each item in the collection.

The following code demonstrates determinging the total length of all LineString geometries within a MultiLineString object:

```
Topobase.Graphic.MultiLineString river =
   new Topobase.Graphic.MultiLineString();
//
// NOT_SHOWN - Assigning LineStrings to "river".
//
double totalLength = 0.0;
foreach (Topobase.Graphic.LineString bank in river)
{
    Topobase.PlaneGeometry.Line2D bank2D = bank.ToLine2D();
    totalLength += bank2D.CalculateLength();
}
```

Moving, Scaling, and Rotating PlaneGeometry Objects

The `Topobase.PlaneGeometry.AffineTransformation` class provides a mechanisms for moving, scaling, and rotating PlaneGeometry objects. You first create a `AffineTransformation` object with the correct series of transformations, and then you call the `Transform()` method which takes in either a `Circle2D`, `Line2D`, `Point2D`, `Polygon2D`, or `Straight2D` object and returns a new transformed object of the same type.

For a simple transformation, you can create a new `AffineTransformation` object by calling one of the static creation methods: `CreateMirror()`, `CreateRotation()`, `CreateScaling()`, and `CreateTranslation()`.

```
// Create transformation that rotates by PI/2 around the origin
// point.

// Use the provided constant to get PI/2.
double angle = Topobase.PlaneGeometry.Vector2D.PiHalf;
Topobase.PlaneGeometry.AffineTransformation rotation;
rotation = Topobase.PlaneGeometry.AffineTransformation.
   CreateRotation(angle);

Topobase.PlaneGeometry.Polygon2D oldPolygon;
//
// NOT SHOWN - creating a polygon object and assigning line seg
ments to it.
//
Topobase.PlaneGeometry.Polygon2D newPolygon;
newPolygon = rotation.Transform(oldPolygon);

// Now newPolygon is a new Polygon2D object that is exactly
// like oldPolygon except it has been rotated PI/2 around the
// origin point.
```

To perform a series of transformations, create an instance of the AffineTransformation object and add each transformation in turn using the `MirrorAt()`, `RotateBy()`, `ScaleBy()`, and `TranslateBy()` methods. The transformations are performed in the order they are added.

```
// Create transformation that first scales the object by
// a factor of 2, and then moves it 10 units along the X axis.

Topobase.PlaneGeometry.AffineTransformation transform1;
transform1 = new Topobase.PlaneGeometry.AffineTransformation();
transform1.ScaleBy(2.0);
Topobase.PlaneGeometry.Vector2D v;
v.X = 10.0;
transform1.TranslateBy(v);

Topobase.PlaneGeometry.Polygon2D oldPolygon;
//
// NOT SHOWN - creating a polygon object and assigning line seg
ments to it.
//
Topobase.PlaneGeometry.Polygon2D newPolygon;
newPolygon = transform1.Transform(oldPolygon);
```

AffineTransformation objects can also be combined using the `CombineWith()` method, and the order of transformations in an AffineTransformation object can be reversed by calling the `Reverse()` method.

SpaceGeometry Namespace

The `Topobase.SpaceGeometry` namespace is the 3D equivalent of PlaneGeometry, and consists of a series of classes that represent simple 3D geometric objects. These classes allow you to perform many kinds of geometric analysis not possible with the objects in the `Topobase.Graphic` namespace.

Be sure to reference *Topobase.SpaceGeometry.dll* in any project that uses this namespace.

Overview of SpaceGeometry

`Topobase.SpaceGeometry` includes classes for many different kinds of simple and compound figures. These classes are all immutable, meaning that they cannot be added to or modified after they have been constructed with the exception of inner rings for polygons. Figures are created by either using the class constructor or by using the static `TryCreate()` method.

Class Name	Description
Cluster	A group of geometries with similar characteristics.
Line3D	A contiguous series of StraightSegment3D objects.
PackedLine3D	A Line3D converted to take up less memory, but which does not allow you to directly access the constituent segments.
Polygon3D	A contiguous series of StraightSegment3D objects forming a closed figure. Holes can be added after the polygon has been constructed.
Sphere3D	A sphere created using a Point3D and a radius.

Class Name	Description
Straight3D	A straight line.
StraightSegment3D	A straight line segment connecting two points in 3D space which can form part of a Line3D or Polygon3D.

There are also two structures representing the basic elements of position and direction.

Structure Name	Description
Point3D	A structure representing a point location in 3D space.
Vector3D	A structure representing a vector in 3D space.

Creating SpaceGeometry Objects From Graphic Objects

Some of the classes in the `Topobase.Graphic` namespace have methods that directly create equivalent SpaceGeometry objects.

Topobase.Grpahic object	Method for creating Topobase.SpaceGeometry object
Point	ToPoint3D()
LinePoint	ToPoint3D()
LineString	ToLine3D()
Polygon	ToLine3D(), ToPolygon3D()

These methods make it easy to perform analysis and mathematical operations on `Topobase.Graphic` objects. The following example determines the 3D area of a `Topobase.Graphic.Polygon` object:

```
// Given a valid Topobase.Graphic.Polygon object named tmpPoly,
// compute the area of this polygon.

Topobase.SpaceGeometry.Polygon3D poly = tmpPoly.ToPolygon3D();
double area = poly.CalculateArea();
```

Those classes that are made up of collections of single geometries (`MultiPoint`, `MultiLine`, `MultiPolygon`) do not have direct methods for creating PlaneGeometry objects. Instead you will need to look through all the items in the collection and call the method for creating those objects for each item in the collection.

The following code demonstrates determinging the total length of all LineString geometries within a `MultiLineString` object:

```
Topobase.Graphic.MultiLineString river =
   new Topobase.Graphic.MultiLineString();
//
// NOT_SHOWN - Assigning LineStrings to "river".
//
double totalLength = 0.0;
foreach (Topobase.Graphic.LineString bank in river)
{
    Topobase.SpaceGeometry.Line3D bank3D = bank.ToLine3D();
    totalLength += bank3D.CalculateLength();
}
```

Creating Graphic Objects From Geometry

Once you have created a PlaneGeometry or SpaceGeometry object, you can create an equivalent Graphic object using the `Topobase.Graphic.GeomFactory` class. First, you get an instance of this class by accessing the static `Default` property - GeomFactory has no new constructor. You then call the `BuildGeometry()` method with a `Point2D`, `Line2D`, `Polygon2D`, `Point3D`, `Line3D`, or `Polygon3D` as the parameter. The function returns a new `Graphic.Point`, `Graphic.Line`, or `Graphic.Polygon` as appropriate.

The following sample code gets the polygon geometry of a feature, converts it into a `Polygon2D`, and scales it by a factor of 2. It then uses that `Polygon2D` to create a `Graphic.Polygon` object using `GeomFactory`.

```
// Get the polygon geometry of a feature named "feature1".
Topobase.Graphic.Polygon tPolygon =
   feature1.Geometry as Topobase.Graphic.Polygon;

Topobase.PlaneGeometry.Polygon2D planePoly =
   tPolygon.ToPolygon2D();

// Use AffineTransformation to scale the polygon.
Topobase.PlaneGeometry.AffineTransformation scale;
scale =
   Topobase.PlaneGeometry.AffineTransformation.CreateScaling(2.0);
planePoly = scale.Transform(planePoly);

// Use GeomFactory to create a new Polygon called "graphicPoly".
Topobase.Graphic.GeomFactory gFactory =
   Topobase.Graphic.GeomFactory.Default;
Topobase.Graphic.Polygon graphicPoly;
graphicPoly = gFactory.BuildGeometry(planePoly);
```

Topologies

There are two main kinds of topologies available in Topobase, logical topologies and area topologies. The collection of all topology objects is obtained from the `TBConnection.Topologies` property.

Topology Related Assemblies

When creating projects that deal with topologies, you will need to include assemblies from the following list:

- *Topobase.LogicalTopology.dll* contains classes to create, modify and analyze logical topologies. It also contains logical topology-specific implementations of the tracing algorithms.
- *Topobase.LogicalToplogy.FeatureRules.dll* contains the classes that can help you automatically update the topology if users add, update, or remove features.
- *Topobase.AreaTopology.dll* contains classes to create, modify and analyze area topologies

- *Topobase.AreaTopology.FeatureRules.dll* contains the classes that can help you automatically update the topology if users add, update, or remove features.
- *Topobase.Tracing.dll* contains the base network tracing classes and interfaces and the basic implementation of tracing algorithms which are used by the tracing functionality of the `Topobase.LogicalTopology` and `Topobase.AreaTopology` namespaces.
- *Topobase.Data.dll* defines all the base metadata classes (topologies, feature classes, features) used throughout the `Topobase.LogicalTopology` and `Topobase.AreaTopology` namespaces.

Logical Topologies

Logical topologies connect features of any feature classes. The features do not need to be spatially connected. A logical topology can connect points with points, or lines with lines, or lines with points, or attribute features with attribute features. For example, a logical topology can represent a waste water network or electrical transmission lines.

A network topology is a subtype of a logical topology that is limited to lines which are geometrically connected to each other at point nodes. Topobase utility models are based on logical topologies that connect points and lines

Topologies are maintained at the client-side using feature rules.

LogicalTopology as Namespace and Class

It is important to note that `LogicalTopology` is both the name of a namespace and the name of a class in the `Topobase.Data` namespace. This means that this class must be fully qualified in your code (that is, always specified as "`Topobase.Data.Doc.Topologies.LogicalTopology`" instead of "`LogicalTopology`"). One remedy is to add the following line to your code:

```
using Topobase.Data.Doc.Topologies as TDDT;
```

You can then just write `TDDT.LogicalTopology` instead of the fully qualified name.

Initializing Logical Topologies

Initializing logical topologies is now done with the `Topobase.LogicalTopology.Maintainance` object. Create a new instance of the maintenance object, use the `InitializeTopology` method, and dispose the maintenance object. One way to make sure the disposal is handled correctly is to use the `using` keyword. Initializing a logical topology is a little more invovled because you need to account for third-party plugins that change how the topology behaves.

```
// Initialize logical topology
using (Topobase.LogicalTopology.Maintainance ma =
 new Topobase.LogicalTopology.Maintainance(myConnection))
{
 // Try to get third-party plugins that change the behavior
 // of the topology.
 Topobase.LogicalTopology.Initialization.
 ITopologyInitializationLogic logic = null;

 Topobase.LogicalTopology.FeatureRules.
 LogicalTopologyRules rules;

 if (Topobase.LogicalTopology.FeatureRules.
 LogicalTopologyRules.TryGetPlugIn(myConnection, out rules))
 {
 logic = rules.GetInitializationLogicFor(topology);
 }

 // Initialize the topology using the found plugins
 ma.InitializeTopology (topology, logic, null);
}
```

Area Topologies

Area topologies are typically used to model surfaces or parcels. An area topology is made up of non-overlapping areas represented by closed polygons made up of line and polyline features. Each area also contains a point feature representing the polygon centroid.

Initializing Area Topologies

Initializing area topologies is now done with the `Topobase.AreaTopology.Maintainance` object. Create a new instance of the maintenance object, use the `InitializeTopology` method, and dispose the maintenance object. One way to make sure the disposal is handled correctly is to use the `using` keyword, as in the following example:

```
// Initialize area topology
using (Topobase.AreaTopology.Maintainance ma =
 new Topobase.AreaTopology.Maintainance(myConnection))
{
 ma.InitializeTopology (topology, null);
}
```

Quick Guide: Creating a Topobase Plugin

4

Introduction

These instructions will guide you step-by-step through the process of creating an Autodesk Topobase plugin that modifies the Autodesk Topobase user interface. For a more detailed description of user interface plugins, see Creating User Interface Plugins (page 105)

C# is used in the following example, but all .NET programming languages can be used to create plugins.

In this example, you will create a new document menu item by performing the following tasks:

- Create a new Visual Studio C# project that will build a class library (a file with a "*.dll*" extension).
- Configure the project so that your plugin and the related files are written to the *bin* folder in the Autodesk Topobase install folder when you build your project.
- Configure the project so that Autodesk Topobase is started when you run your plugin from Visual Studio.
- Configure the project with the class references your plugin class needs.
- Review the assembly properties to verify that the default values are valid.
- Add code to the default class.

- Create a Topobase plugin definition file (identified by the *.tbp* extension) so that Autodesk Topobase knows about your plugin.
- Test your plugin.

The intent of the plugin is to add an menu item to the context menu within the Document Explorer in the Topobase task pane. Selecting the menu item causes a message box to be displayed with the text Hello World.

IMPORTANT Testing this plugin requires a workspace containing feature classes based on the Point feature class type. A suitable workspace for testing the plugin is the TB2009_LM_102 Land Management demonstration workspace. The creation of this workspace is described in the *Autodesk Topobase Installation and Configuration Guide*. Another suitable workspace is the TBSAMPLE workspace described in Sample 01 - Create Structure (page 149).

Creating a New Project

This procedure describes how to create a new C# class library project. All Autodesk Topobase plugin types are based on class libraries.

1. Start Microsoft Visual Studio 2005.
2. Click File ➤ New ➤ Project. The New Project dialog box is displayed.
3. Click the node labeled Visual C# in the New Project dialog box in the Project Types pane
4. In the Templates pane click the item labeled Class Library.
5. In the text box labeled Name type “MyTopobasePluginProject”.
6. Set the path to the project folder by using the Location selection box or by clicking the Browse... button.
7. Click OK to close the New Project dialog box and create a new blank solution.

The Solution Explorer pane is now populated with a solution node as root with a project subnode. The project node has two subnodes: My Project and Class1.cs. The editor pane is populated with a tab labeled Class1.cs, and the contents of this tab is a skeleton class declaration:

```
class Class1
{
}
```

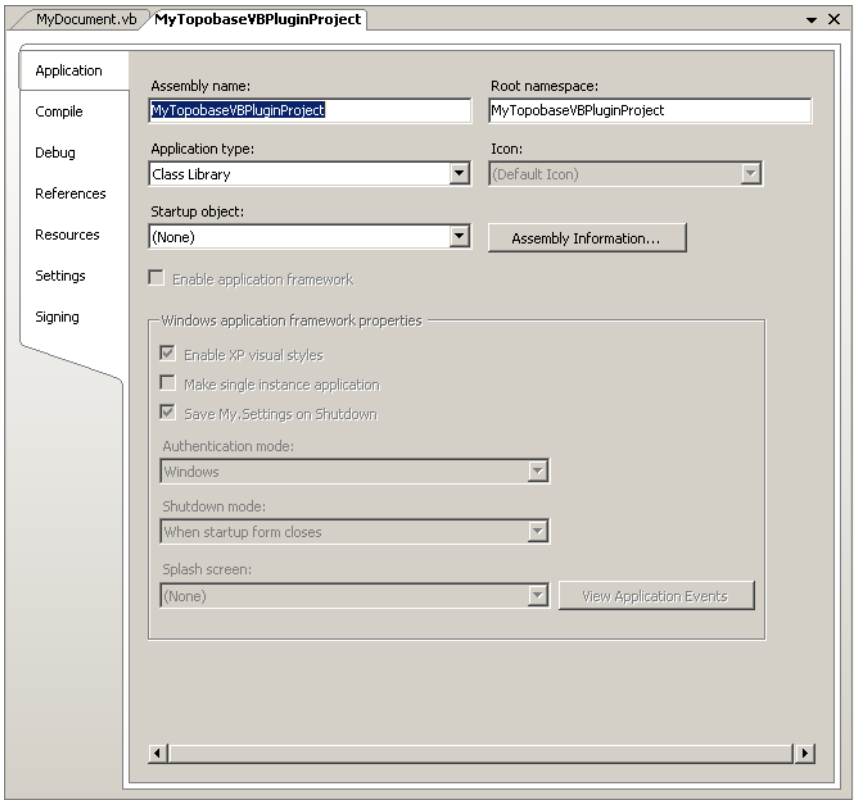
Accessing the Project Properties

To configure your project for use with Topobase, you must edit the Visual Studio project properties. This brief procedure describes how to display the required panel and shows an example image. Specific information about configuring the properties starts with Configuring the Project to Write the Build Outputs to the Topobase Bin Folder (page 54).

To display the Visual Studio project properties:

- In the pane labeled Solution Explorer, right-click the project node and select Properties from the popup menu.

The result is that the project properties are displayed in a tab in the editor pane. The tab label is the name of the project. The contents of this tab is a set of vertical tabs on the left and a display area on the right. Depending on your Visual Studio configuration, the vertical tabs are labeled from top to bottom: Application, Compile, Debug, References, Resources, Settings, and Signing (and possibly, Security, and Publish). Clicking a vertical tab causes the display of related properties in the display area on the right.



MyTopobasePluginProject - Visual Studio Project Properties

After changing any of the properties, you will notice that the properties tab label now has an asterisk appended to it as does the label of the vertical tab where you made the change. Save the changes by placing the cursor over the properties tab label, right-click to display the popup menu and select Save Selected Items from the menu. The result is that the asterisks are removed from the properties tab and vertical tab labels.

Configuring the Project to Write the Build Outputs to the Topobase Bin Folder

This procedure will cause the project build outputs to be written to the Autodesk Topobase bin folder. The build outputs are a `.dll`, `.pdb`, and `.xml` file.

1. Display the project properties.
2. In the properties tab click the vertical tab labeled Compile.
3. If required, change configuration to All Configurations.
4. Click the Browse... button to the right of the text box labeled Build Output Path.
5. In the Select Output Path dialog box browse to the location of the Autodesk Topobase bin folder. The path is typically *C:\Program Files\Autodesk Topobase Client 2009\bin*. Click Open. This causes the path that you selected to be written to the text box labeled Build Output Path.
6. Save the properties.

Configuring the Project to Run Topobase from Visual Studio

This procedure will cause Autodesk Topobase to run automatically when you start debugging your project in Visual Studio. This will allow you to debug the plugin while Topobase is running. You can place breakpoints in the code and use the "Edit and Continue" feature of Visual Studio .NET to make changes to your code without exiting Topobase.

1. Display the project properties.
2. In the properties tab click the vertical tab labeled Debug.
3. If required, change configuration to All Configurations.
4. In the set of radio buttons whose group label is Start Action click the radio button to the left of the label Start External Program.
5. Click the Browse... button to the right of the text area for this radio button.
6. In the Select File dialog box, browse to the folder containing the Autodesk Topobase executable. This path is typically *C:\Program Files\Autodesk Topobase Client 2009*. Select *acad.exe* and click Open.
7. Under the group label Start Options in the text box labeled Working Directory type the location of *acad.exe*. For this exercise this location is *C:\Program Files\Autodesk Topobase Client 2009*.
8. Save the properties.

Adding References To Your Project

You must add a number of Autodesk Topobase class references to your project.

1. In the Solution Explorer pane place the cursor over the project node and right-click to display the popup menu. Select Add Reference... from the menu.
2. In the Add Reference dialog box, click the Browse tab.
3. Browse to the location of the Autodesk Topobase Client 2009 bin folder. Holding down the Ctrl key, select the following:
   - Topobase.Data.dll
   - Topobase.Forms.dll
   - Topobase.Graphic.dll
   - Topobase.Map.dll

   NOTE Not all plugins require all these references. Some other kinds of plugins have their own requirements. For example, workflows require a reference to Topobase.Workflows.
4. Click OK.
5. In the editor pane note that the properties tab label has an asterisk appended to it again. Click the vertical tab labeled References. Note that Topobase.Data.dll, Topobase.Forms.dll, Topobase.Graphic.dll, and Topobase.Map.dll have all been added to the list box labeled References:.
6. In the list box labeled References do the following for each of these *.dll*'s:
   1. Select it. The result is that the Properties pane is populated with the properties associated with that *.dll*.
   2. In the Properties pane select the property labeled Copy Local. This causes a select control to appear in the box containing the value for this property.

      NOTE This also causes help text to be displayed for that property. The help text reads: Indicates Whether The Reference Will Be Copied To The Output Directory.
   3. Use the select control to change the value of this property to False.

4 In the Properties pane select the property labeled Specific Version. This causes a select control to appear in the box containing the value for this property.

NOTE This also causes help text to be displayed for that property. The help text reads: Indicates Whether This Reference Is To A Specific Version Of An Assembly.

---

5 Use the select control to change the value of this property to False.

7 Save the properties.

Inspecting the Assembly Information

This procedure describes how to the inspect the assembly information generated for this project.

1 In the project properties tab click the vertical tab labeled Application.

2 Click the button labeled Assembly Information. This causes the Assembly Information dialog box to be displayed.

3 Inspect the information and click OK.

Adding Code To The Default Class

When you created the project, you caused a default class called `Class1` to be created in a file called *Class1.cs*. Change the name of the class to `MyDocument`. In the Solution Explorer pane right-click the node labeled Class1.cs to display its popup menu. Select Rename from the menu. The display of the class node's label is changed to a text box. Type MyDocument.cs in the text box. Click the tab in the editor pane containing the skeleton class definition. The tab label is changed to MyDocument.cs, and the class name in the skeleton class declaration is changed to `MyDocument`.

1 Modify the class declaration so that it looks like the following:

```
public class MyDocument : Topobase.Forms.DocumentPlugIn
{
    public Topobase.Forms.MenuItem myMenuItem1 =
       new Topobase.Forms.MenuItem();
}
```

The public declaration of the myMenuItem1 object allows Topobase to communicate menu events in the user interface to the plugin.

2 Add the following code to the class. This will override the event handler which is called when the Autodesk Topobase document menus are first initialized.

```
public override void OnInitMenus
   (object sender, Topobase.Forms.Events.MenusEventArgs e)
{
}
```

3 Add a menu item to the popup menu by adding the following lines of code into the body of the new `OnInitMenus` subroutine (Note that this menu item will *only* be available to point feature class items in the task pane):

```
// Obtain the context menu.
Topobase.Forms.Menu pointMenu =
  e.Menus.Item(Topobase.Data.FeatureClassType.Point);

// Add the menu item to the context menu.
pointMenu.MenuItems.Add(myMenuItem1, "My Menu Item");

myMenuItem1.Click += new Topobase.Forms.Events.
  MenuItemClickEventHandler(MyMenuItem1_Click);
```

4 Create a handler for menu events by adding the following lines of code below the `OnInitMenus` subroutine:

```
private void MyMenuItem1_Click
  (object sender, Topobase.Forms.Events.MenuItemClickEventArgs
e)
{
}
```

5 Display a message box when the menu item is clicked. Add the following line of code in the body of new new `MyMenuItem1_Click` subroutine:

```
this.Application.MessageBox("Hello World");
```

6 Save the class file. After the above steps, the class should look like the following:

```
public class MyDocument : Topobase.Forms.DocumentPlugIn
{
public Topobase.Forms.MenuItem myMenuItem =
   new Topobase.Forms.MenuItem();
```

```
public override void OnInitMenus
   (object sender, Topobase.Forms.Events.MenusEventArgs e)
{
    // Obtain the context menu.
    Topobase.Forms.Menu pointMenu =
      e.Menus.Item(Topobase.Data.FeatureClassType.Point);

    // Add the menu item to the context menu.
    pointMenu.MenuItems.Add(myMenuItem1, "My Menu Item");

    myMenuItem1.Click += new Topobase.Forms.Events.
      MenuItemClickEventHandler(MyMenuItem1_Click);
}


private void MyMenuItem1_Click
  (object sender, Topobase.Forms.Events.MenuItemClickEventArgs
e)
{
    this.Application.MessageBox("Hello World");
}

}
```

Creating a Topobase Plugin Definition File

When Autodesk Topobase starts, it reads the plugin definition files in the *bin* folder to determine what plugins are available.

Use a text editor to create a file with a *.tbp* extension. For this example, name the file *MyTopobasePlugInProject.tbp*. Save it in the project directory with your source code. Add it to the list of files in the Visual Studio Solution Explorer dialog box. Bring up the properties of the file and make sure that the "Copy to Output Directory" property is set to "Copy Always". This will copy the *.tbp* file to the Autodesk Topobase Client 2009 *bin* folder with the *.dll* whenever you build your project. The contents of the file are as follows:

```
<?xml version="1.0" encoding="utf-8"?>

<PlugIn>

  <Default
```

```
    AssemblyName="MyTopobasePlugInProject.dll"
    Namespace="MyTopobasePlugInProject"
    DocumentKey=""
    MapName=""
    Priority="100"
    ExecutionTargetWeb="True"
    ExecutionTargetDesktop="True"
  />
<DocumentPlugIn ClassName="MyDocument"/>
</PlugIn>
```

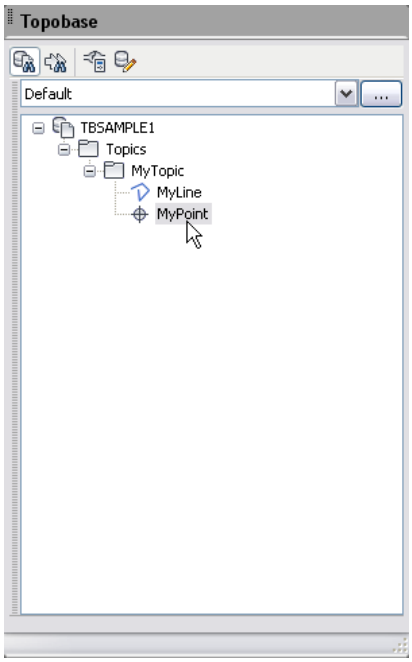
TIP If you copy the above text and paste it into your *MyTopobasePluginProject.tbp* file, ensure that the pasted quotation marks are ASCII-encoded.

NOTE The AssemblyName attribute value is `MyTopobasePlugInProject.dll`. By default the *.dll* name is the same as the project name.

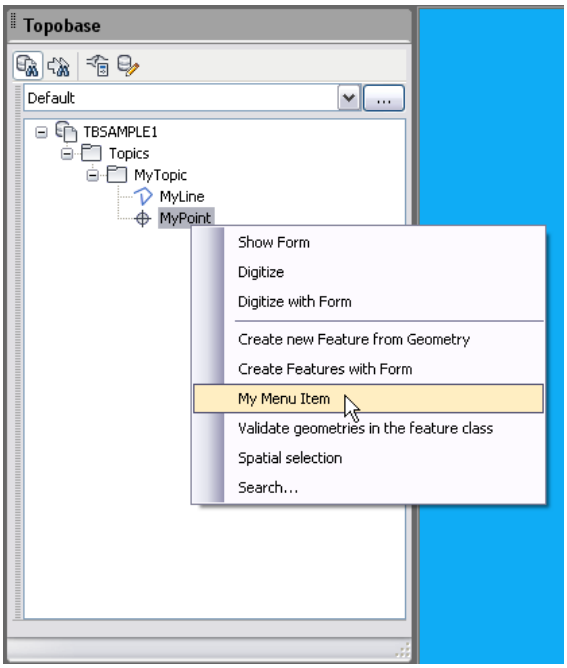
Testing the Plugin

If the plugin project is configured to start Autodesk Topobase (see Configuring the Project to Run Topobase from Visual Studio (page 55)), then you can test and debug your plugin from within Visual Studio. You can place breakpoints in the code and use the "Edit and Continue" feature of Visual Studio .NET to make changes to your code without exiting Topobase.

1. In Visual Studio build the solution and then select the Start Debugging command. This will start the Autodesk Topobase Client application.
2. Log into Autodesk Topobase and select a workspace that contains some point feature classes.
3. Open the "Point" folder in the document explorer so that you see the following hierarchy:



4 Place the cursor over any leaf node based on the Point feature class type, and right-click to display the popup menu. You should see My Menu Item in the menu, as shown in the following graphic:



MyTopobasePluginProject - My first MenuItem

5 Select My Menu Item to activate the message box with the text Hello World.

Creating Application Modules

5

A vertical application module serves as the basis for documents created in the Topobase Administrator. An application module defines the data model (topics, tables, attributes, topologies, and so on) and provides tools for manipulating the data model (workflows, plugins, option pages). This chapter will describe how to build an application module based on the example provided by Sample 117 - Simple Water Module (page 272) and how to create plugins and other tools, either as part of the application module or as stand-alone Topobase extensions.

Creating the Data Model

About Data Models

A vertical application module often needs to create or update a data model, add domain values to a model, create or update a topology, add workflow definitions, or perform other administrative tasks. A type of plugin called the “structure update” plugin is used to perform these actions. If your application module does not need to perform these tasks, then do not include a structure update plugin in the project.

All data model changes that can be done manually in the Topobase Administrator can be done in an automated way using the structure update mechanism. The advantage with using the programmatic structure update mechanism is that the changes are codified, documented, automated and portable between Oracle instances. For example, in moving from a test environment to a production environment, the same data model changes performed in the test environment must be exactly replicated in the production environment. If these updates have been performed manually, they would need to be documented, and repeated exactly with no missing or extra steps. The

structure update API and structure update plugin allows these steps to be automated, and hence reproduced with fidelity in the target system. A similar and even more compelling argument applies when deploying Topobase customizations between branch offices of a company or if the customization is part of a product for resale.

Creating the Structure Update Plugin

The first step in defining a data model is the creation of a structure update plugin. The structure update plugin provides an interface to the data model definitions and provides a mechanism for updating the data model when the data model is modified. Structure update plugins are derived from the abstract class `Topobase.Update.DocumentStructureUpdatePlugIn`.

Structure update plugins need to override the following abstract properties and methods:

- **Commentary** - Property returning a string that describes the data model. The return value can be null.
- **DataModelCode** - Property returning a DMCodeNumber structure containing the identification of the data model. The structure consists of three integers, the company code, the module code, and a subcode. Pick a number between 3 and 99 for your company code and use it for all data models you create. Company codes 1, 2, and 42 are reserved by Autodesk. The module code usually serves to identify the type of module (such as wastewater, gas, electric and so on). The subcode is typically used to identify the localization. You can return a null value for the DMCodeNumber structure, but you should then override the ModuleName property and return a meaningful value.

  NOTE There are no central institutions for confirming that your company code is unique.

- **DataModelCodeText** - Property returning a string that gives the name of the data model. The return value can be null.
- **Image16** - Property returning a System.Drawing.Image object containing a 16x16 pixel icon. This image is used to help identify the module, and is displayed in the Topobase Administrator Document Info dialog next to the module name. The return value can be null, in which case no icon is displayed.

(To see the Document Info dialog, first load a document in Topobase Administrator. Click on the Document ➤ Data Model tree item. Right click on the root of the data model and select “Document Info”.)

- **GetVersions()** - Method that returns a `Topobase.Update.StructureUpdateVersions` collection containing all of the update version modules based on the `Topobase.Update.StructureUpdateVersionBase` abstract class. The `Topobase.Update.StructureUpdateVersions` collection is built using the `StructureUpdateVersions.CollectAllStructureUpdateClasses()` static method.

This is an example of a structure update module:

```
public class StructureUpdatePlugIn:
 Topobase.Update.DocumentStructureUpdatePlugIn
{
 public override string Commentary
 {
 get
 {
 return Resources.StructureUpdateCommentary;
 }
 }

 public override DMCodeNumber DataModelCode
 {
 get
 {
 // Return an appropriate data model code if
 // this is the updater for an application module.
 return (new DMCodeNumber(
 int.Parse(Resources.DataModelCompanyCode),
 int.Parse(Resources.DataModelCode),
 int.Parse(Resources.DataModelSubCode)));
 }
 }

 public override string DataModelCodeText
 {
 get
 {
 // Return a name for this data model
 return Resources.ModuleName;
 }
 }

 public override System.Drawing.Image Image16
 {
 get
 {
 return null;
 }
 }

 // Gets all data model versions of this data model.
```

```
 public override Topobase.Update.StructureUpdateVersions GetVer
sions()
 {
 // Gets all versions of this assembly through reflection.
 StructureUpdateVersions versions =
 StructureUpdateVersions.CollectAllStructureUpdateClasses(
 this.GetType(),
 this.TBConnection,
 this.Application);

 return versions;
 }
}
```

Creating an Update Version Module

The data model is defined in a series of update version classes, which are derived from the `Topobase.Update.StructureUpdateVersionBase` abstract class. Each time you change the data model you will add a new `StructureUpdateVersionBase`-derived class which codifies the changes between the previous version and the latest version. The initial creation of the data model uses the same method - you are just "updating" from an empty version 0.0.0 of the data model.

If your update version module is updating a separate, already existing module, such as the Water data model (data model code 2.5.0) you would still start at 0.0.0, but conditionally load your module based on the presence and version of the Water data model. Stand-alone update version modules can be conditionally loaded by adding a "DMCode" element to the .tbp file for your structure update plugin:

```
<DocumentStructureUpdatePlugIn Priority="90" DMCode="2.5.0"
Namespace="MyModifiedDataModel" ClassName="MyStructureUpdatePlugIn"
```

Version update classes need to override the following properties and methods:

- **Version** - Property returning a `Topobase.Data.VersionNumber` structure holding the version of the changes to the data model. Each version update class added to the module should be given a sequential version number. The order of the version numbers determines the order the structure updates are applied. You cannot use version "0.0.0", which is used to indicate the state before any data model information was added.

- **Commentary** - Property returning a string describing the nature of the update to the data model.
- **UpdateCondition** - Property returning a boolean value indicating whether the update should proceed or not in the current state. In most cases you should return true unless you can determine that the update has already been applied, in which case you should return false. If you return true without checking if the update has been applied, the first argument to your model update calls in `UpdateStructure()` should probably be `StructureUpdateExceptionHandling.Skip` to avoid throwing exceptions if domains, classes, or other elements already exist.
- **UpdateStructure()** - Method that actually defines the data model. This is done by adding or modifying topics, feature classes, labels, domain tables, domain entries, attributes, relations, rule bases, and workflows as needed using methods of the `Topobase.Update.Structure` class.

This is an example of an update version module that creates a data model consisting of two topics and two feature classes.

```
public class Version10000:
 Topobase.Update.StructureUpdateVersionBase
{
 // Gets the version number of this update
 public override Topobase.Data.VersionNumber Version
 {
 get
 {
 return new VersionNumber(1, 0, 0);
 }
 }

 // Gets a commentary which may be shown in the update dialog.
 public override string Commentary
 {
 get
 {
 return Resources.Version10000Description;
 }
 }

 // Gets whether this update should be executed
 // or not by checking to see if items we are
 // going to add already exist.
 public override bool UpdateCondition
 {
 get
 {
 if (TBConnection.FeatureClasses.Contains("DEMO_PIPE"))
 return false;
 else
 return true;
 }
 }

 // This method updates the data model.
 public override void UpdateStructure()
 {
 AddTopic("Pipe", DataModelResources.topicPIPE);
 AddTopic("Point", DataModelResources.topicPOINT);

 AddFeatureClass(
 "Pipe",
```

```
 "DEMO_PIPE",
 DataModelResources.fcDEMO_PIPE,
 FeatureClassType.Attribute);
 AddFeatureClass(
 "Point",
 "DEMO_VALVE",
 DataModelResources.fcDEMO_VALVE,
 FeatureClassType.Attribute);
 }
}
```

Adding Changes to the Data Model

After a data model has been created, you can no longer modify the existing version update modules, as this would cause incompatibility with any document using the old data model. Instead, you add a new update version module which contains all the additions and modifications to the previous data model. Make sure that the overridden `Version` property returns a version number greater than the any previous version number.

The following version update module will add two attributes to the DEMO_PIPE feature class that was created in class Version10000:

```
public class Version10001:
 Topobase.Update.StructureUpdateVersionBase
{
 // Gets the version number of this update
 public override Topobase.Data.VersionNumber Version
 {
 get
 {
 return new VersionNumber(1, 0, 1);
 }
 }

 // Gets a commentary which may be shown in the update dialog.
 public override string Commentary
 {
 get
 {
 return Resources.Version10001Description;
 }
 }

 // Gets whether this update should be executed
 // or not by checking to see if items we are
 // going to add already exist.
 public override bool UpdateCondition
 {
 get
 {
 FeatureClass fc;
 fc = TBConnection.FeatureClasses["DEMO_PIPE"];
 if (!fc.Attributes.Contains("DATE_ACQUIRED") ||
 !fc.Attributes.Contains("DATE_INSTALLATION"))
 return true;
 else
 return false;
 }
 }

 // This method updates the data model.
 public override void UpdateStructure()
 {
 FeatureClass fc;
 Topobase.Data.Attribute attr;
```

```
fc = TBConnection.FeatureClasses["DEMO_PIPE"];

if (!fc.Attributes.Contains("DATE_ACQUIRED"))
{
attr = new Topobase.Data.Attribute(
fc,
"DATE_ACQUIRED",
Topobase.Data.DataType.Date);
attr.Caption =
DataModelResources.attrDATE_ACQUIREDCaption;
attr.Description =
DataModelResources.attrDATE_ACQUIREDDescription;
fc.Attributes.Add(attr);
}

if (!fc.Attributes.Contains("DATE_INSTALLATION"))
{
attr = new Topobase.Data.Attribute(
fc,
"DATE_INSTALLATION",
Topobase.Data.DataType.Date);
attr.Caption =
DataModelResources.attrDATE_INSTALLATIONCaption;
attr.Description =
DataModelResources.attrDATE_INSTALLATIONDescription;
fc.Attributes.Add(attr);
}

SendUpdateInfoMessage(
DataModelResources.UpdateInfoMessage,
fc.Caption);
}
}
```

Structure update versions are atomic revisions to the data model. These updates are unidirectional - the version can only be advanced from lower versions to higher versions. These updates are permanent. The data model in Oracle is being adjusted to the current version of the module. There is no reverse process to 'undo' version updates.

Data Model Elements

Topics

Topics are collections of feature classes, and are used to organize data elements in the Topobase Data Model pane. Topics do not imply any relationship between the feature classes contained within them.

The following line creates a “Pipe” topic that will be used to contain all the pipe-related feature classes:

```
Topic pipeTopic = AddTopic("Pipe", "Pipe Related Classes");
```

Topics can also contain other topics to create a hierarchical data structure. The following line creates a “Special” topic that is located under the parent “Pipe” topic:

```
// Get the name string from a resource file.
AddTopic("Special", "Pipe Related Classes", pipeTopic);
```

Feature Classes

A feature class represents a type of Topobase object, and corresponds to a single table within the database. Feature classes are created using the `Structure.AddFeatureClass()` method.

NOTE Data model tables should have prefix describing the purpose (for example, “LM_” for land management). Sample 117 uses the prefix “DEMO_”.

The following line creates a “DEMO_PIPE” feature class, which will be displayed as part of the “Pipe” topic.

```
AddFeatureClass(
 "Pipe",
 "DEMO_PIPE",
 DataModelResources.fcDEMO_PIPE,
 FeatureClassType.Attribute);
```

Labels

Labels are a kind of feature class that is attached to a parent feature class which displays a text string at a specified point. Labels are created with the `Structure.AddLabelFeatureClass()` method.

NOTE Label feature classes automatically have suffix "_TBL" attached to them.

The following line adds a label feature class named "DEMO_PIPE_TBL" to the "DEMO_PIPE" feature class:

```
AddLabelFeatureClass(
 "DEMO_PIPE",
 "DEMO_PIPE_TBL",
 DataModelResources.lblFcDEMO_PIPE_TBL);
```

Attributes

Each feature class type contains a set of predefined attributes (that is, columns in the database table). User defined attributes can also be added to feature classes. Attributes are added or modified using the `Attributes` collection in a feature class object.

The following code first gets a reference to the "DEMO_PIPE" feature class. It then checks to make sure the attribute has not already been added. If it has not, then it creates a new attribute named "PIPE_LENGTH", a numeric value, and then adds it to the feature class.

```
FeatureClass fc;
Topobase.Data.Attribute attr;

fc = TBConnection.FeatureClasses["DEMO_PIPE"];
if (fc == null)
 return;

if (!fc.Attributes.Contains("PIPE_LENGTH"))
{
 attr = new Topobase.Data.Attribute(
 fc,
 "PIPE_LENGTH",
 Topobase.Data.DataType.Number);
 attr.Precision = 20;
 attr.Scale = 8;
 attr.UnitID = TBConnection.GetUnit(MeasuringUnitType.Length);
 attr.Caption = DataModelResources.attrPIPE_LENGTHCaption;
 attr.Description = DataModelResources.attrPIPE_LENGTHDesc;
 fc.Attributes.Add(attr);
}
```

Attribute Relations

An attribute relation defines a relationship between attributes in different tables. It can define the association cardinality between attributes and/or an action to take with the child attribute when something happens to the parent attribute. Attribute relations are set using the `StructureAddAttributeRelation()` method.

The following line creates a relation between the parent “FID” attribute of the DEMO_PIPE feature class and the child “FID_FEATURE” of the DEMO_MAINTENANCE feature class. When a feature of type Pipe is deleted, the maintenance features associated to it are deleted as well.

```
AddAttributeRelation(
 "DEMO_MAINTENANCE",
 "FID_FEATURE",
 "DEMO_PIPE",
 "FID",
 RelationType.DeleteChild);
```

Domains

Domains

User defined attributes can be created using common data types, such as numbers or strings. They can also be limited to a specified enumeration of values. A list of possible values such an attribute can have is called a "domain." Each domain corresponds to a single table in the database. Domains are created using the `Structure.AddDomainTable()` method.

NOTE Domain tables automatically have suffix "_TBD" attached to them.

This line creates a domain named "DEMO_MAINTAINENANCE_TYPE_TBD":

```
AddDomainTable(
 "DEMO_MAINTENANCE_TYPE_TBD",
 "DEMO MAINTENANCE TYPE");
```

Domain Values

Values within a domain are added individually. The following code adds three new values to the DEMO_MAINTAINENANCE_TYPE_TBD domain, "u", "o", and "tbd":

```
AddDomainEntry("DEMO_MAINTENANCE_TYPE_TBD",
 1,
 "unknown",
 "u",
 "description",
 true);

AddDomainEntry("DEMO_MAINTENANCE_TYPE_TBD",
 10000,
 "other",
 "o",
 "description"
 true);

AddDomainEntry("DEMO_MAINTENANCE_TYPE_TBD",
 10001,
 "to be determined",
 "tbd",
 "description"
 true);
```

Domain Relations

A domain relationship is used to set a particular attribute to use the defined domain. This line sets the ID_MAINTENANCE_TYPE attribute of the DEMO_MAINTENANCE feature class to use the DEMO_MAINTENANCE_TYPE_TBD domain:

```
AddRelationToDomain(
 "DEMO_MAINTENANCE_TYPE_TBD",
 "DEMO_MAINTENANCE",
 "ID_MAINTENANCE_TYPE");
```

Now a ID_MAINTENANCE_TYPE attribute can hold one of three values, “u”, “o”, and “tbd”.

Utility Model

A utility model is a system of organizing feature classes for utility networks. It contains the relationship between the geometry and the attribute feature classes and adds network geometry.

The following code creates a utility model where the feature classes DEMO_VALVE and DEMO_HYDRANT can be used as nodes in the network and the feature class DEMO_PIPE is used for the connecting lines:

```
// Make an array of all the feature class names that act
// as nodes.
string[] attrPoints = new string[]
 { "DEMO_VALVE", "DEMO_HYDRANT" };
// Make an array of all the feature class names that act
// as lines.
string[] attrLines = new string[] { "DEMO_PIPE" };
CreateUtility("DEMO", "DEMO_LINE", attrLines, "DEMO_POINT",
 attrPoints, 0.01d, "DEMO", true);
```

Client-Side Feature Rules

Another aspect of the data model is the client-side feature rule. Feature rules define the business rules of the data model, performing consistency checks, dependency checks, data correction, and many other tasks. A number of feature rules are predefined in Topobase, but you can create more as needed. Rules are stored in the TB_RULE_DEF table.

The following example accesses the predefined rule named "CreateStartEndNode" and assigns it to the DEMO_LINE feature class, passing the name of the DEMO_VALVE feature class as a parameter.

```
RuleDef ruleDef = TBConnection.RuleDefs
 ["CreateStartEndNode",
 "Topobase.Utilities.dll",
 "Topobase.Utilities.FeatureRules.UtilityLineFeatureRules"];

FeatureClass fc = TBConnection.FeatureClasses["DEMO_LINE"];
TBConnection.RuleBases.Add(ruleDef, fc, "DEMO_VALVE");
```

Workflows

Adding Workflows

Workflows are a kind of plugin that guides the user through a series of steps that analyze or modify the displayed geometry. Workflows can be created and added to the client environment in many different ways, but you may wish that data model-specific workflows be loaded when the rest of the data model is created. The following code can be used to load a workflow class that is defined within the application module project:

```
AcquisitionWorkflows acquisitionWorkflowInstance =
 new AcquisitionWorkflows();

AddOrUpdateWorkflow(
 acquisitionWorkflowInstance.NetworkPipeCreation,
 Resources.ResourceManager);
```

Organizing

A flat listing of large numbers of workflows may be difficult to use. You can organize workflows in a hierarchical structure by using the `Structure.AddOrUpdateWorkflow()` method to create nodes in the workflow tree to contain other workflows. To create such a node, set the parameter that would normally contain the workflow script to null.

```
Workflow parentNode = AddOrUpdateWorkflow(
 "DEMO ACQUISITION ParentNode",
 Resources.WorkflowNameAcquisition,
 "Workflow",
 0, // Priority
 null, true);
```

You can then add workflows to this node by calling an override of `AddOrUpdateWorkflow()` that takes the parent node as a parameter:

```
// Create the code block that loads a workflow
// plugin library.
string codeBlock = "Sub Run";
codeBlock += Environment.NewLine;
codeBlock += "Me.RunMethod(\"workflow.dll\",";
codeBlock += "\"ProjectName.ClassName\", \"EntryPointFunction\")";

codeBlock += Environment.NewLine;
codeBlock += "End Sub"
codeBlock += Environment.NewLine;

// Create the workflow as a child of the parentNode
// created earlier.
AddOrUpdateWorkflow(
 "DEMO ACQUISITION workflow",
 Resources.WorkflowNameAcquisition,
 parentNode,
 null,
 0, // Priority
 codeBlock);
```

Further Information

For more information about workflows, see: Creating Workflows (page 84)

Installing an Application With a Structure Update Plugin

.TBP File

The .tbp file for module should have a line like the following for the structure update plugin:

```
<DocumentStructureUpdatePlugIn Priority="90" DMCode=""
Namespace="CSSample117.DataModel" ClassName="StructureUpdatePlugIn"
 />
```

Creating a Stand-Alone Topobase Extension

Just as with all parts of a vertical application module, you can create a stand-alone componet out of a `DocumentStructureUpdatePlugIn` module.

Such extensions can be used to create portions of a data model that can be used in different workspaces. Dimension, Plot, Templates, COGO are all examples of extensions.

When creating a `DocumentStructureUpdatePlugIn` extension, be sure to override the `DocumentStructureUpdatePlugIn.Category` method and return the correct category type, as in the following example:

```
/// <summary>
/// Defines the Plot as an extension
/// </summary>
/// <returns>CategoryType.Extension</returns>
public override CategoryType Category()
{
 return CategoryType.Extension;
}
```

Adding Feature Rules

About Feature Rules

Feature rules are short procedures that are triggered whenever changes are made to the database. They are used to enforce business rules within a document by automatically setting attributes, rejecting incorrect new features, repairing errors, performing housekeeping actions, and other tasks. For example, feature rules might move child labels whenever a feature is moved, or split a vertex when a new pipe it attached to it at a point besides one of the end points.

Creating Feature Rules

Creating the FeatureRulePlugin Module

Feature rule plugins are classes based on the `FeatureRulePlugIn` abstract class which list and operate the feature rules for the application module. Feature rule plugins need to overwrite the `GetRules()` method to return an array of descriptions of all rules, each of which includes a reference to a function within the class.

```
public override Topobase.Data.FeatureRules.FeatureRuleDef[]
 GetRules()
{
 FeatureRuleDef[] result = new FeatureRuleDef[1];

 // Describe a feature rule that will fire before any new
 // feature is inserted.
 result[0] = new FeatureRuleDef(
 100,
 "UpdateDateUserAttributes_BI",
 Resources.FeatureRule_UpdateDateUserAttributes_BI,
 RuleTriggerEvent.BeforeInsert,
 new InsertUpdateFeatureRuleDelegate
 (UpdateDateUserAttributes_BI),
 "UpdateModificationHistory",
 new VersionNumber("1.00.00"));

 return result;
}
```

Creating the Rules

The rule consists of a function which takes in the feature class, feature, and parameters being checked. It then performs whatever action is required by the rule, such as modifying an attribute of the feature. It also returns a boolean value indicating whether the feature passes the rule or not. The following feature rule updates the date created and username attributes:

```
private bool UpdateDateUserAttributes_BI(FeatureClass fc, Feature
 feature, params string[] parameters)
{
 if (fc.Attributes.Contains(DataModelResources.attrDateCreated))
 {
 feature.Attributes[DataModelResources.attrDateCreated]
 .Value = DateTime.Now;
 }
 if (fc.Attributes.Contains(DataModelResources.attrUserCreated))
 {
 feature.Attributes[DataModelResources.attrUserCreated]
 .Value = userName;
 }

 return true;
}
```

Rule Priorities

When defining a rule, the first parameter of the `FeatureRuleDef` constructor is the priority of the rule - an integer which indicates in which order the rule should be activated relative to other rules. A poorly chosen value can make it and other rules not work correctly. The recommended value for the priority parameter depends on the time the rule is activated (that is, what `RuleTriggerParameter` value is set for the fourth parameter of the constructor).

For “Before Insert”, “Before Update”, “Before Insert Or Update”, and “Before Delete” rules:

Priority value	Use
10-19	Used to cancel operations when conditions do not depend on other rules. **Example: Cancel all updates of point features.**
30-39	Used to change attributes of features being inserted/updated. **Example: Set orientation of points.**
50-59	Used to cancel operations when conditions depend on other rules. **Example: Cancel all updates of points with invalid orientation.**

Priority value	Use
70-79	Used for all other tasks. **Example: Find all lines connected to the point and store them in a variable.**
90-99	Used to make changes to the database. **It is recommended that you do not use this priority, and instead create an after-operation rule instead.**

For “After Insert”, “After Update”, “After Insert Or Update”, and “After Delete” rules:

Priority value	Use
110-119	(Reserved for future use.)
130-139	Used to delete features in the database. **Example: Delete lines connected to point.**
150-159	Used to update features in the database. **Example: Move lines connected to point.**
170-179	Used to insert features into the database. **Example: Split a line.**
190-199	Used for all other tasks. **Example: Regenerate graphics.**

Installation

The *.tbp* file for module should have a line like the following for the feature rules plugin:

```
<FeatureRulePlugIn ClassName="FeatureRulesClassname" />
```

Creating Workflows

About Workflows

A workflow is a program that guides the user through a complicated task. When a workflow is started, the user is presented with a user interface prompt, such as a message box or controls in the Topobase task pane, directing them through a series of steps such as setting workflow parameters, selecting features for analysis, or digitizing new features. There are two ways to create a workflow program - writing Visual Basic script in Topobase Administrator or by creating a workflow plugin using Visual Studio .NET.

Workflow Scripts in Topobase Administrator

The Topobase Administrator includes a tab for creating new workflows within a workspace document. A textbox allows the user to type in the workflow code in the Visual Basic .NET language.

To create a new blank workflow, bring up the Topobase Administrator and load the workspace. Select “Workflows” in the tree view under the document you want to add the workflow to. In the Workflow Administrator page, select the parent workflow in the tree view and click Create. Type the name of the new workflow. You will now see a new entry for the workflow and a simple template for adding your workflow code. You can adjust the name, display name, and icon as needed. When you are done creating the workflow, be sure to select the Save button.

The entry point for a workflow script is the `Run` subroutine. You can create other subroutines and functions, but a script must contain at least the `Run` subroutine.

There are a number of limitations when working with workflow scripts. The code window is limited in size, and the script interpreter provides only the most basic of features. There are no debugging features. You can only access a subset of the Visual Basic .NET language. Because of these issues, workflow scripts are only useful for the simplest of workflows.

This is a sample workflow script. It prompts the user for a name for a new valve, has the user digitize a new WA_VALVE feature, and sets the name of the feature to the string the user gave earlier. It then opens the feature dialog box to allow the user to type in new attribute values.

```
Sub Run
 Dim name As String
 name = InputBox("Input name of new valve:", "Digitize Valve")

 If (not string.IsNullOrEmpty(name)) Then
 me.Digitize("WA_VALVE")
 ' "0" is that always the latest item that was digitized
 me.Features.List.Item(0).Attributes.Item("NAME_NUMBER").Value =
name
 me.Features.Update()
 me.OpenDialog()
 End If
End Sub
```

Use of the "Me" Object

Workflow scripts are derived from the `Topobase.Workflows.ScriptClass` abstract class. By using the `Me` object in the script, you can gain access to the many base objects and useful functions provided by `ScriptClass`.

These are the objects provided by the `Me` object. With these you can access all of the main features of the Topobase environment and document data.

- **Application** - This object contains information about the current working environment, the current user, and the open workspace and documents. It is a reference to the base application object derived from the `Topobase.Forms.Application` abstract class.
- **Document** - This objects allows you to interact with the document that this workflow is part of, including bringing up dialog boxes or closing the document. It is a reference to one of the documents in the current workspace, an object derived from the `Topobase.Forms.Document` abstract class.
- **Features** - This object provides methods for accessing and modifying the list of features that were just recently digitized. It is an instance of the `Topobase.Workflows.Features` class.

```
' Digitize a new pipe and then display a message box with
' the feature ID of the newly created feature.
Me.Digitize("DEMO_PIPE")
Me.MessageBox(Me.Features.ListItem(0).Attrib
utes.Item("FID").Value)
```

The following are some of the useful methods provided by the `Me` object:

- **ShowMessage()** - Hides any explorer currently displayed in the Topobase task pane and displays a string message inside. When the workflow ends, the Topobase task pane reverts to its previous state.
- **MessageBox()** - Displays a standard message box with a single "Ok" button and a single string message with an optional title. Processing of the workflow pauses until the message box is dismissed.
- **InputBox()** - Displays a standard input box allowing the user to input a string value. Processing of the workflow pauses until the input box is dismissed.
- **Digitize()** - Allows the user to add features of the specified feature class to the document.
- **RunMethod()** - Runs a method from an external .dll library. The most important use of RunMethod is to launch workflow plugins.

Workflow Plugin

A workflow plugin is a class library created in Visual Studio .NET. It must contain a class based on `Workflows.WorkflowPlugIn`, but otherwise you have freedom in creating its design.

Create a Workflow Plugin Module

Workflow plugins are classes derived from the `Workflows.WorkflowPlugIn` abstract parent class.

```
class AcquisitionWorkflows : Topobase.Workflows.WorkflowPlugIn
{
}
```

Creating the Entry Point

Workflows are accessed by an entry point - a public function with no parameters and no return value. You can use any function name. Instead of parameters, the instance of the `WorkflowPlugIn`-derived class has many properties and methods for getting information about the current document and the state of the environment. That instance can be accessed using the `this` keyword in C# or `Me` keyword in Visual Basic .NET.

Generally, the entry point should perform three actions. First, it should check if the current state of Topobase and of the document will allow the workflow to work correctly. Second, it should request input from the user. Third, it should update the database based on the user's input.

The following sample entry point demonstrates a particular kind of workflow which creates an interface in the workflow pane. This entry point checks the current state of Topobase, creates the user interface elements for initializing the workflow into the workflow pane, and then adds an event to the user interface elements to start the interactive part of the workflow when the initialization phase is complete. The event (`CreateNetworkPipe_Closed`) will contain the code for updating the database.

```
public void NetworkPipeCreation()
{
 // Determine if Topobase is in the correct state for the
 // workflow to be meaningful.
 // Check if the utility is loaded.
 if (this.Utility == null)
 return;

 // Also check if the map is connected to the document.
 if (this.Document.Map.IsConnected(true))
 return;

 // Create the container in the workflow pane with Ok and
 // Cancel buttons.
 IWorkflowContainer container = WorkflowContainerFactory.Create(
 this.WorkflowSupport,
 ContainerButtons.OkCancel,
 Resources.ModuleName,
 false);

 // Add a control to the container that asks if the user
 // wants to set a reference for the features they are about
 // to digitize.
 FeatureClass fc = Document.Connection.
 FeatureClasses["DEMO_PIPE"];
 RefFeatureData refData = new RefFeatureData(
 this.WorkflowSupport, "refData", fc,
 string.Format(
 CultureInfo.CurrentCulture,
 Resources.ChooseReferenceOf,
 fc.Caption)
 );

 IWorkflowUserControl refControl =
 WorkflowUserControlFactory.Create(refData);

 container.AddUserControl(refControl);

 // Make the container visible.
 this.WorkflowSupport.WorkflowExplorerFlyIn.
 ShowWorkflowContainer(container);

 // Add an event for when the user selectes the Ok or
```

```
    // Cancel buttons on the container.
    container.Closed += new EventHandler(CreateNetworkPipe_Closed);
  }
```

Create a Workflow Pane UI

The defining characteristic of a workflow is that it guides the user through a process by using user interface messages and controls. Topobase provides many different systems of interacting with the user, depending on the need of the workflow. You may need only the simplest message displays, or the full flexibility and options of a Windows user control, or the compatibility between different versions of the Topobase Client that the Topobase controls can provide.

Displaying Simple Messages

Message Box

One way to give messages to the user is to display a modal message box, which the user can dismiss and then continue with the workflow process. Message boxes can be displayed with the `Application.MsgBox()` method.

```
this.Application.MsgBox("Hello");
```

This is the simplest way to present a message to the user. The `Application.MessageBox()` method performs a similar task, but also gives you the options of setting the message box title and using a mono-spaced font.

```
// Use a message of "Hello", a title of "Title", and the
// standard font.
this.Application.MessageBox("Hello", "Title", false);
```

ShowMessage

Another way to give the user a message is to use the `WorkflowSupport.WorkflowExplorerFlyIn.ShowMessage()` method. This will hide any explorer in the Topobase task pane and display a string centered within that area. This message is modeless, so it will continue to be displayed as the user performs workflow tasks, such as digitizing features.

```
this.WorkflowSupport.WorkflowExplorerFlyIn.
 ShowMessage("Start Digitizing. Press ESC when done.");
```

The Topobase task pane will return to its previous state once the workflow ends. You can also dismiss the message by using the `HideUserControls()` method.

```
this.WorkflowSupport.WorkflowExplorerFlyIn.HideUserControls();
```

Displaying a Windows User Control

A workflow user interface can be implemented by creating a Windows user control. When a workflow is started, the user control is resized and rendered in the Topobase task pane. A user control allows you to more exactly specify the location and organization of controls and gives you the ability to use a wide variety of Visual Studio toolbox components. However, workflows using Windows user controls are not compatible with the Topobase Web Client.

To create a workflow using a user control for the workflow's user interface, first add a Windows user control to the project by selecting the Project ➤ Add User Control... menu item in Microsoft Visual Studio. This user control will contain the user interface for setting up the workflow. Add any controls and labels you need. Be sure to design you control so it works correctly when resized to fit within the Topobase task pane.

Next, add the workflow operations. This will usually be in response to a control event (such as the click event of an Ok button). At the end of the workflow operation hide the user control and return the Topobase user interface to its previous state.

```
private void ButtonOk_Click(object sender, System.EventArgs e)
{
 // Add code that conducts the Workflow operation here.
[..]

 // Hide the user control and return the task
 // pane to its previous state.
 workflowSupport.WorkflowExplorerFlyIn.HideUserControls();
}
```

Next, create the WorkflowPlugin class with an entry point. In the entry point, create and display the Windows user control.

```
public void EntryPoint()
{

[..]

 // Create a new instance of the Windows user control
 // named "MyWorkflowUserControl".
 MyWorkflowUserControl form =
 new MyWorkflowUserControl(this.Workflow);

 // Display it in the Topobase task pane.
 this.Workflow.WorkflowExplorerFlyIn.ShowControl(form);

[..]

}
```

See Sample 85 for a demonstration of this technique.

Using WorkflowContainerFactory

A workflow user interface can also be implemented by directly creating Topobase workflow controls in the Topobase task pane using an object based on the `IWorkflowContainer` interface (`GeneralWorkflowContainerDesktop` or `GeneralWorkflowContainerWeb`) created using the `WorkflowContainerFactory` static class. Workflows based on these container objects are compatible with the Topobase Desktop Client and, unlike workflows using Widows user controls, the Topobase Web Client as well. While the choice of controls and their placement is limited compared to Windows user controls, this also gives you access to Topobase-specific controls. See Topobase Workflow Controls (page 92) for more information about these controls.

In the entry point, make a container object that contains the `IWorkflowContainer` interface. This can be done by calling `WorkflowContainerFactory.Create()`, which will return an `IWorkflowContainer` interface to the new container object, either a `GeneralWorkflowContainerDesktop` or a `GeneralWorkflowContainerWeb` depending on whether the user is in the Topobase Desktop Client or the Topobase Web Client.

NOTE If you know the workflow will only be used in one type of client, you can directly create a new GeneralWorkflowContainerDesktop or GeneralWorkflowContainerWeb object.

When creating this container, you specify some features of its behavior, such as if it has a “Close” button, or “Ok” and “Cancel” buttons.

```
IWorkflowContainer container = WorkflowContainerFactory.Create(
 this.WorkflowSupport,
 ContainerButtons.OkCancel,
 Resources.ModuleName,
 false);
```

When your workflow has finished, be sure to clear the Topobase task pane and return it to its previous state with the following code:

```
this.WorkflowSupport.WorkflowExplorerFlyIn.HideUserControls();
```

Sample 85, Sample 116, and Sample 117 demonstrate this technique.

Topobase Workflow Controls

The following controls can be placed within a container based on `IWorkflowContainer`.

CheckBox Control

A check box control creates a series of check boxes positioned vertically within a group box. The data object contains an array of data items, one for each check box within the control.

Creating a CheckBox Control

```
// Create the control's data object.
CheckBoxData checkData = new CheckBoxData
 (this.Workflow,
 "CheckBoxKey",
 "Title", /* title of the group box */
 "MyWorkflowTest",
 "CheckBoxKey");
// Create the individual check box designs. Create an array of
// DataItems,one for each check box. The following code
// creates one check box.
 CheckBoxData.DataItem[] cbItems = new CheckBoxData.DataItem[1];
// Define the specifics of a single check box. It will have the
// text "Check 1", and it will be checked.
CheckBoxData.DataItem cbItem1 = new CheckBoxData.DataItem
 ("Check 1", "Check1Tag", true);
// Assign the newly created specifics to a place within the array.
cbItems.SetValue(cbItem1, 0);
// Assign the array to the data container.
checkData.AvailableItems = cbItems;
// Create a control containing the group box and check boxes as
// defined in the control data object.
CheckBoxControlDesktop checkControl =
 new CheckBoxControlDesktop(checkData);
// Add the control to the workflow container.
container.AddUserControl(checkControl);
```

Getting User Input From CheckBox

The status of a check box is determined by looping through all the data object data items, determining which data item represents the check box you are interested in, and testing its `CheckState` property.

```
// Get the list of all control data objects.
WorkflowDataList dataList = container.DataList;
// Retrieve the check box data object from the list of
// control data objects.
CheckBoxData checkData = dataList["CheckBoxKey"] as CheckBoxData;
foreach (CheckBoxData.DataItem check in checkData.CheckedItems)
{
 if (check.CheckState == true)
 {
 // Do something if the checkbox is checked.
 }
}
```

RadioButton Control

A radio button control creates a series of related radio buttons positioned vertically within a group box. The data object contains an array of data items, one for each radio button within the control.

Creating a RadioButton Control

```
// Create the control's data object.
RadioButtonData radioData = new RadioButtonData
 (this.Workflow,
 "RadioKey",
 "Radio Title",
 Properties.Resources.ModuleName,
 "RadioKey");
// Create the individual radio button designs. Create an array
// of DataItems, one for each radio button. The following
// code creates one radio button.
RadioButtonData.DataItem[] rdItems =
 new RadioButtonData.DataItem[1];
// Define the specifics of a single radio button. It will have the
// text "Radio 1", and it will be selected.
RadioButtonData.DataItem rdItem1 =
 new RadioButtonData.DataItem("Radio 1", "Radio1Tag", true);
// Assign the newly created specifics to a place within the array.
rdItems.SetValue(rdItem1, 0);
// Assign the array to the data container.
radioData.AvailableItems = rdItems;
// Create a control containing the group box and radio buttons
// as defined in the control data object.
RadioButtonControlDesktop radioControl =
 new RadioButtonControlDesktop(radioData);
// Add the control to the workflow container.
container.AddUserControl(radioControl);
```

Getting User Input From RadioButton

The selected radio button is determined by checking the `CheckedItem` property of the data object. `CheckedItem` is a reference to a single `RadioButtonData.DataItem` object.

```
// Get the list of all control data objects.
WorkflowDataList dataList = container.DataList;
// Check which radiobutton option the user selected.
RadioButtonData RadioSelections =
 dataList["RadioKey"] as RadioButtonData;
// We will look at the Tag property to see if the checked
// radio button is the one we are interested in.
bool IsSelected =
  RadioSelections.CheckedItem.Tag.Equals("Radio1Tag");
```

ComboBox Control

The ComboBox control creates a drop-down list combo box with a label above it. The data object contains an array of data items, one for each item within the drop-down list.

Creating a ComboBox Control

```
// Create the control's data object.
ComboBoxData comboData = new ComboBoxData
 (this.Workflow,
 "ComboBoxKey",
 "Combo Title",
 Properties.Resources.ModuleName,
 "ComboBoxKey");
// Create the lines that populate the drop-down list.  Create an
// array of DataItems, one for each line.  The following code
// creates two lines.
ComboBoxData.DataItem[] cmbItems = new ComboBoxData.DataItem[2];
ComboBoxData.DataItem cmbItem1 =
 new ComboBoxData.DataItem("List Item 1", "tag1");
cmbItems.SetValue(cmbItem1, 0);
ComboBoxData.DataItem cmbItem2 =
 new ComboBoxData.DataItem("List Item 2", "tag2");
cmbItems.SetValue(cmbItem2, 1);
// Assign the array to the data container.
comboData.AvailableItems = cmbItems;
// Create a control containing the combobox and label as
// defined in the control data object.
ComboBoxControlDesktop comboControl =
 new ComboBoxControlDesktop(comboData);
// Add the control to the workflow container.
container.AddUserControl(comboControl);
```

Getting User Input From ComboBox

The text of the selected item is determined by checking the `SelectedItem` property of the combo box data object.

```
// Get the list of all control data objects.
WorkflowDataList dataList = container.DataList;
// Get the data object of the combobox control from the list of
// all control data objects.
ComboBoxData comboData = dataList["ComboBoxKey"] as ComboBoxData;
// Display the text of the selected item.
System.Windows.Forms.MessageBox.Show
 ("Combobox selected item: " + comboData.SelectedItem.Text);
```

RefFeatureControl Control

The RefFeatureControl control is a combo box that allows the user to select a reference for the features to be digitized. It also contains a pre-defined label prompting the user to select the reference type.

Creating a RefFeatureControl

```
// Get a reference to the feature class we are going to be digit
izing.
FeatureClass fc = Document.Connection.FeatureClasses["DEMO_PIPE"];

// Create the setup information for the RefFeatureControl.
RefFeatureData refData = new RefFeatureData(
 this.WorkflowSupport,
 "refData",
 fc,
 string.Format(CultureInfo.CurrentCulture,
 Resources.ChooseReferenceOf,
 fc.Caption));

// Create the control generically, so that it works in both
// desktop and web versions.
IWorkflowUserControl refControl =
 WorkflowUserControlFactory.Create(refData);

// Add the control to the pane.
container.AddUserControl(refControl);
```

Getting User Input From RefFeatureControl

```
WorkflowDataList dataList = container.DataList;

RefFeatureData refData = dataList["refData"] as RefFeatureData;
```

ChoseFeatureClass Control

The ChooseFeatureClass control creates a list box populated with a list of all feature class names. The list box allows selection of multiple lines. It also contains a pre-defined label prompting the user to select one of the feature classes.

Creating a ChooseFeatureClass Control

```
// Create the control's data object.
ChooseFeatureClassData chooseClassData = new ChooseFeatureClassData
 (this.Workflow,
 "ChooseClassKey",
 this.Document.Connection.FeatureClasses,
 false,
 false,
 true,
 Properties.Resources.ModuleName,
 "ChooseClassKey");
// All options to set for the ChooseFeatureClass control are
// set within the constructor for the control data.
// Create a control containing a populated list
// as definied by the control data object.
ChooseFeatureClassControlDesktop chooseClassControl =
 new ChooseFeatureClassControlDesktop(chooseClassData);
// Add the control to the workflow container.
container.AddUserControl(chooseClassControl);
```

Getting User Input From ChooseFeatureClass

An array of `FeatureClass` objects is contained in the `SelectedFeatureClasses` property of the `ChooseFeatureClassData` data object. These `FeatureClass` objects represent those items in the list box which are currently selected.

```
WorkflowDataList dataList = container.DataList;
ChooseFeatureClassData chooseFeatureClassData =
 dataList["ChooseClassKey"] as ChooseFeatureClassData;
// Make sure the user selected a feature class.
if (chooseFeatureClassData.SelectedFeatureClasses.Count == 0)
{
 // The user didn't select any feature class. Simply hide the
 // user controls and return so the workflow is terminated.
 Workflow.WorkflowExplorerFlyIn.HideUserControls();
 return;
}
// Get the first FeatureClass object from the list of
// selected feature class names.
FeatureClass selectedFeatureClass =
  chooseFeatureClassData.SelectedFeatureClasses[0];
```

ChoseFeature Control

The ChooseFeature control consists of a button, group box, and a pre-defined label with the number of currently selected features. Pressing the button allows the user to select features within the drawing.

Creating a ChooseClass Control

```
// Create the control's data object.
ChooseFeatureData choosefeatureData = new ChooseFeatureData
 (this.Workflow,
 "ChooseKey",
 true, /* Allow multiple selection */
 true); /* Need at least one feature selected */
// Create a control as defined in the control data object.
ChooseFeatureControlDesktop choosefeatureControl =
 new ChooseFeatureControlDesktop(choosefeatureData);
// Add the control to the workflow container.
container.AddUserControl(choosefeatureControl);
```

Acquisition Workflows - Basics of Digitizing

Using DigitizeFeatures

The `WorkflowSupport.Document.Map` object contains a method for digitizing features. Calling `DigitizeFeatures()` allows the user to create any number of the specified type of features. The features created will be returned as a `FeatureList`. The prompt to display while digitizing is set by one of the parameters. The workflow application is paused until the user finishes digitizing the new features or presses escape to cancel the operation. If the user cancels the operation, then `DigitizeFeatures()` returns a valid list containing zero elements.

```
FeatureList digitized;
digitized = this.WorkflowSupport.Document.Map.DigitizeFeatures(
 featureClass,
 featureClass.Type,
 "Please digitize a " + featureClass.Caption,
 null,
 true);
```

Digitizing Utility Features

If you need to digitize node or line features into a utility network (such as a water pipe or electrical network), then you would use the digitize methods of the `UtilityFeatreClass` class.

`UtilityFeatureClass` represents a feature class that is part of a utility. Objects of this type are created by the static factory class `UtilityFeatureClassFactory`. `UtilityFeatureClass` contains the methods `Digitize()` and `DigitizeMutiple()` which allow the user to create new utility features within the document.

No prompt is displayed to the user. You should use `ShowMessage` (or some other method) to tell the user what state they are in and what action they should take.

This sample creates an instance of the `UtilityFeatureClass` based on the existing DEMO_PIPE feature class using the `UtilityFeatureClassFactory`. It then loops, allowing the user to create any number of individual DEMO_PIPE features, which represent lines within the water utility network.

```
FeatureClass fc;
fc = Document.Connection.FeatureClasses["DEMO_PIPE"];

UtilityFeatureClass pipeUfc;
pipeUfc UtilityFeatureClassFactory.GetInstance(fc);
ArrayList pipes = new ArrayList();

bool newPipe = true;
while (newPipe)
{
 // Give the user a prompt
 this.WorkflowSupport.WorkflowExplorerFlyIn.
 ShowMessage("Digitize Pipe.  Press ESC to exit.");
 UtilityLine pipe = pipeUfc.Digitize(Document) as UtilityLine;

 if (pipe == null)
 {
 newPipe = false;
 break;
 }
 pipes.Add(pipe);
}
```

The `UtilityFeatureClass` class contains many other methods that can be useful in workflows:

- **ConvertFeature** - Allows the user select a feature in the document which is converted to a utility feature and returned.
- **HighlightFeatures** - Takes an array of utility features and highlights them in the display. It can optionally zoom the display to show all the hightlighted features.
- **SelectSingleFeatureInGraphic** - Allows the user to select a utility feature in the document. It can optionally return which line segment of the feature was selected.

Analysis Workflows - Basics of Analysis

Selecting Points and Regions

You can prompt the user to select points, rectangles, or polygons within the document by using methods of the `WorkflowSupport.Document.Map` object.

`GetPoint()` will give the user a prompt that is displayed as a tooltip by the cursor and in the command window. The method returns the point location in the map where the user clicked. The workflow application is paused until the user selects a point or presses escape to cancel the operation.

```
// The prompt will say "Select Point"
Topobase.Graphic.Point pt;
pt = this.WorkflowSupport.Document.Map.GetPoint("Select Point");
```

`GetRectangle()` will give the user a prompt that is displayed as a tooltip by the cursor and in the command window. The method returns the polygon region in the map defined by two clicks from the user. The workflow application is paused until the user selects the rectangle or presses escape to cancel the operation.

```
Topobase.Graphic.Polygon rect;
rect = this.WorkflowSupport.Document.Map.
 GetRectangle("First Corner", "Second Corner");
```

`GetPolygon()` will also give the user a prompt that is displayed as a tooltip by the cursor and in the command window. The method returns the polygon region in the map defined by at least three clicks from the user. The workflow application is paused until the user selects the region or presses escape to cancel the operation.

```
Topobase.Graphic.Polygon poly;
poly = this.WorkflowSupport.Document.Map.
 GetPolygon("Select Polygon");
```

Selecting Features in the Document

One way for the user to select features in the document is to call the `GetFeatures()` method of the `WorkflowSupport.Document.Map` object. `GetFeatures()` allows the user to select any number of features (either one at a time or through selection windows and crossing windows) until they press the enter key. The features selected will be returned as a `FeatureList`. You

cannot set the prompt - it will automatically display the text “Select Entities”. The workflow application is paused until the user finishes selecting features or presses escape to cancel the operation.

```
FeatureList features;
features = this.WorkflowSupport.Document.Map.GetFeatures();
```

Installation

Adding Workflows Automatically

If your workflow class is part of a vertical application, you can add your workflow to the workspace while creating your data model. This is done using the `AddOrUpdateWorkflow()` method.

There are many different overridden versions of the `AddOrUpdateWorkflow` method, but there are two general forms. Most versions have you pass the same information as you would manually enter in the Workflow Administrator page of the Topobase Administrator. The following example creates a new workflow by passing the workflow name, the caption to be displayed in Topobase, the path to the icon file, the priority value, the script code, and a boolean value indicating that this workflow should not be modified by the user.

```
AddOrUpdateWorkflow(
 "DEMO ACQUISITION ParentNode",
 Resources.WorkflowNameAcquisition,
 NodeWorkflowPictureName,
 0,
 null, true);
```

NOTE This particular sample passes a null string for the script code - this means that this entry in the list of workflows does not actually do anything, but it can serve as the parent of other workflows for organizational purposes.

Another overridden form of the `AddOrUpdateWorkflow` method takes a reference to the workflow entry point method and a reference to the resource that contains the caption string. Attached to the entry point method is a series of function attributes defining the icon file path, priority, and other settings that would normally be passed to `AddOrUpdateWorkflow` as parameters.

```
// In a structure update module where the data model
// is defined.
public override void UpdateStructure()
{
 AcquisitionWorkflows acquisitionWorkflowInstance =
 new AcquisitionWorkflows();

 AddOrUpdateWorkflow(
 acquisitionWorkflowInstance.NetworkPipeCreation,
 Resources.ResourceManager);
}



// Inside the workflow plugin class.

// Set the function attributes describing the workflow.
[WorkflowMethodDescription(
 AcquisitionWorkflows.WorkflowNameCreateNetworkPipe,
 "WorkflowNameNetworkPipeCreation",
 "WaterPipeCreation",
 1, // Priority
 ParentWorkflowName = "DEMO ACQUISITION ParentNode",
 IsSystem = false)]
public void EntryPoint()
{
[...]
```

NOTE The benefit of this is to locate all the workflow-related settings within the workflow class itself for easier editing. A vertical application might contain a large number of very complicated structure update classes, and it can be hard to locate the correct AddOrUpdateWorkflow method to make changes.

Adding Workflows Manually Using Administrator

If your workflow is located in a stand-alone class library file (that is, a standalone *.dll*), you will need to add your workflow information manually into the Topobase Administrator using the following steps:

1. Build the project. Place the plugin library into the Topobase Client bin directory. The associated .tbp file must also be copied to the Topobase Client bin directory.

2 Start Topobase Administrator.

3 Load your workspace. Select the folder labeled Workflows in the left pane under the appropriate document.

4 In the Workflow Administrator page, click Create.

5 In the Create Workflow dialog box enter a name for the workspace and click Create.

6 Create a script in the Script Code edit field that uses the `Me.RunMethod` function to call the entry point function in your plugin library. The first parameter is the filename of the workflow library. The second parameter is the namespace and class name of the workflow. The third parameter is the name of the entry point method.

```
Sub Run
 Me.RunMethod("workflow.dll",
 "ProjectName.ClassName", "EntryPointFunction")
End Sub
```

7 Click the Save button.

Organizing Workflows

You can organizeworkflows in a hierarchy by creating blank workflows using the `AddOrUpdateWorkflow` method while passing a null string for the Script Code parameter. These blank workflows can act as parents for your functional workflows. Assign the child workflows by using one of the overloads of the `AddOrUpdateWorkflow` method that takes a parent workflow object as a parameter.

Creating User Interface Plugins

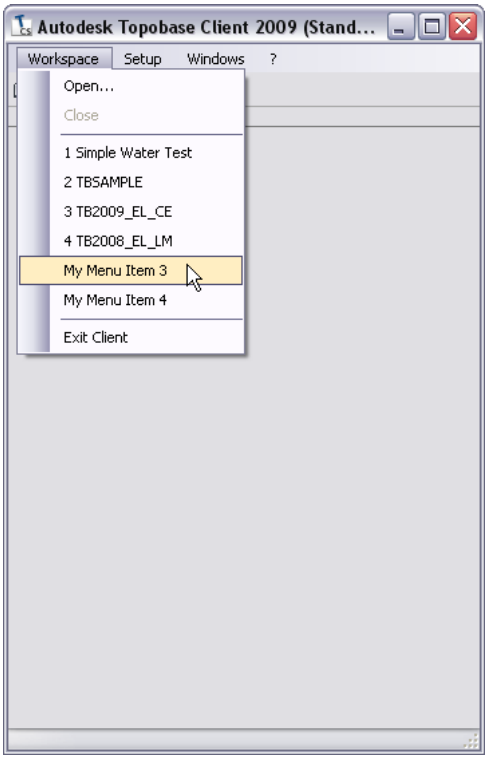
User interface plugins are small applications that provide functionality to the Topobase user, usually by creating a new control integrated into some part of the Topobase client user interface. There are many different kinds of plugins which correspond to the different user interface elements of Topobase and to the different levels of information (application-wide tasks, tasks dealing with a single document, or tasks dealing with a single table).

Creating an Application Plugin

About Application Plugins

Application plugins are a mechanism for performing application-wide operations. They may run without a user interface until an application event fires, or they may modify the application menu bar or add application toolbars and toolbar buttons to the Topobase task pane.

NOTE The application menu bar is only visible in the stand-alone Client application or in the desktop Client application when no workspace is loaded. If a workspace is loaded in the desktop Client, the menu bar is not visible.



Application menu of the Topobase task pane in the Topobase standalone client showing two custom entries from an application plugin.

These plugins are derived from the `Topobase.Forms.ApplicationPlugIn` class:

```
public class MyApplication : Topobase.Forms.ApplicationPlugIn
{
}
```

Responding to Application Events

The `ApplicationPlugIn` class contains a number of virtual functions which you override to respond to application events. These functions include:

- `OnPreLoad` - Triggered first when the plugin is loaded and executed.
- `OnLoad` - Triggered during the initialization process when the plugin is loaded and executed.
- `OnInitComplete` - Triggered after the initialization process is complete.
- `OnInitMenus` - Override this to create application menu items.
- `OnUnload` - Triggered when the plugin is unloaded. Not triggered when the Topobase Client is closed.

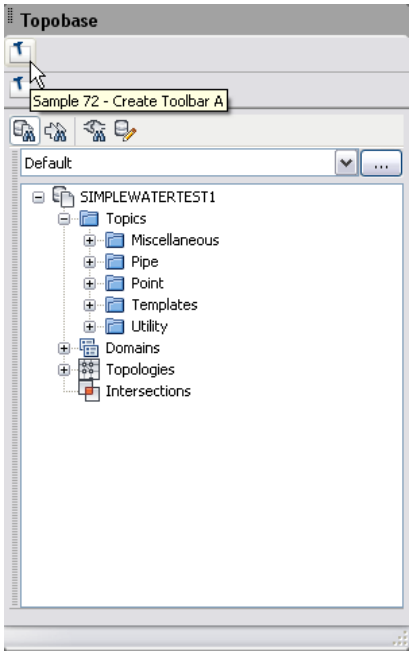
In the following example, an application plugin named "MyApplication" displays a dialog box when the plugin is first loaded:

```
public class MyApplication : Topobase.Forms.ApplicationPlugIn
{
 public override void OnLoad(object sender, EventArgs e)
 {
 base.OnLoad(sender, e);

 // Bring up the dialog box (defined elsewhere).
 Form1 theForm = new Form1();
 theForm.Show();
 }
}
```

Adding Custom Toolbars and Toolbar Buttons

By default, no application toolbars are shown in the Topobase task pane. Application plugins can create new toolbars by passing a string containing the new toolbar name to the `Item` property of the `ToolBars` collection. These

toolbars can be shared between different plugins if they use the same toolbar name.



Two application toolbars are added by Sample 72, an application plugin.

To create a toolbar and toolbar button, create a global public variable of type `Topobase.Forms.ToolBarButton`. Override the `ApplicationPlugIn` class's `OnInitToolBars` event, create a new application toolbar, and insert a new button using the `Add` method of the toolbar's `Buttons` collection. You can then create an event handler for the new button and can respond to button events.

```
public Topobase.Forms.ToolBarButton myToolbarButton =
 new Topobase.Forms.ToolBarButton();

public override void OnInitToolBars
 (object sender, Topobase.Forms.Events.ToolBarsEventArgs e)
{
 // Create a custom toolbar.
 Topobase.Forms.ToolBar customToolbar;
 customToolbar = e.ToolBars.Item("Custom Toolbar");

 // Add a new toolbar button to the application toolbar
 // using the "Topobase" icon and "My Toolbar Button"
 // as the tooltip text.
 customToolbar.Buttons.Add
 (myToolbarButton,
 "Topobase",
 "My Toolbar Button");

 // Add event handler for the button.
 myToolbarButton.Click +=
 new EventHandler(myToolbarButton_Click);
}


private void myToolbarButton_Click
 (object sender, System.EventArgs e)
{
 // Perform the plugin action here.
 this.Application.MessageBox("Toolbar button was clicked");
}
```

Adding Menu Items

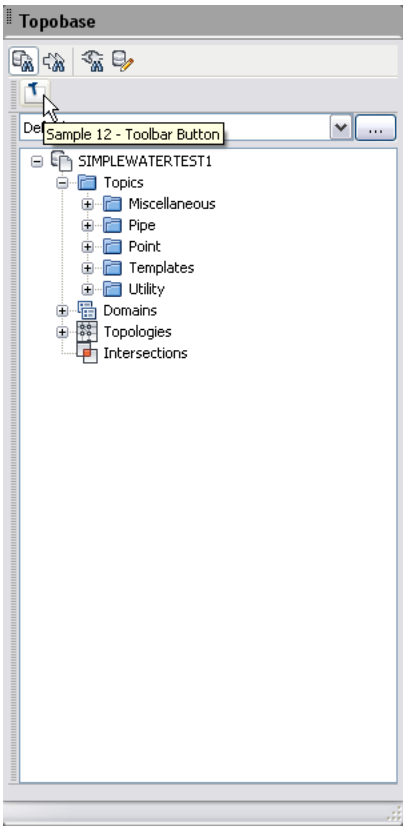
To add a menu item, create a public variable of type `Topobase.Forms.MenuItem`. Override the `ApplicationPlugIn` class's `OnInitMenus` method (which is triggered when the menu initialization is taking place), obtain a reference to the application menu, and insert a new menu item using the `Add` method of the menu's `MenuItems` collection. You can then create an event handler for the new menu item and can respond to menu events.

```
public Topobase.Forms.MenuItem myMenu =
 new Topobase.Forms.MenuItem();

public override void OnInitMenus
 (object sender, Topobase.Forms.Events.MenusEventArgs e)
{
 // Get the application's project menu.
 Topobase.Forms.Menu vlobMenu =
 e.Menus.Item(Topobase.Forms.ApplicationMenuType.Workspace);

 // Add the new menu item.
 vlobMenu.MenuItems.Add(myMenu, "My Menu Item");

 // Bind the events.
 myMenu.Click += new Topobase.Forms.Events.
 MenuItemClickEventHandler(myMenu_Click);
}


private void myMenu_Click
 (object sender, Topobase.Forms.Events.MenuItemClickEventArgs e)
{
 // Perform the plugin action here.
 this.Application.MessageBox("Menu item was clicked");
}
```

Creating a Document Plugin

About Document Plugins

Document plugins are a mechanism for performing document level operations. They can set up event handlers for document level notifications, or they can add new features to the user interface. For example, a document plugin can set up an event handler for job state transitions, and perform client side validation or reports prior to a job being committed to the “Live” state. Document plugins can also modify the user interface by adding items to the document toolbar or menu items to the context menus within the Topobase task pane:



Document-specific toolbar in the Topobase task pane.

Such plugins are derived from the `Topobase.Forms.DocumentPlugIn` class:

```
public class MyDocument : Topobase.Forms.DocumentPlugIn
{
}
```

Creating Custom Toolbars and Toolbar Icons

By default, no document toolbars that can be modified by document plugins are shown in the Topobase task pane. Document plugins can create new toolbars by passing a string containing the new toolbar name to the `Item` property of the `ToolBars` collection. These toolbars can be shared between different plugins if they use the same toolbar name.

To create a toolbar and toolbar button, create a global public variable of type `Topobase.Forms.ToolBarButton`. Override the `DocumentPlugIn` class's `OnInitToolBars` event, create a new toolbar, and insert a new button using the `Add` method of the toolbar's `Buttons` collection. You can then create an event handler for the new button and can respond to button events.

```
public Topobase.Forms.ToolBarButton myToolbarButton =
 new Topobase.Forms.ToolBarButton();

public override void OnInitToolBars
 (object sender, Topobase.Forms.Events.ToolBarsEventArgs e)
{
 // Create a custom toolbar.
 Topobase.Forms.ToolBar customToolbar;
 customToolbar = e.ToolBars.Item("Custom Toolbar");

 // Add a new toolbar button to the document toolbar
 // using the "Topobase" icon and "My Toolbar Button"
 // as the tooltip text.
 mainToolbar.Buttons.Add
 (myToolbarButton, "Topobase", "My Toolbar Button");

 myToolbarButton1.Click +=
 new EventHandler(myToolbarButton_Click);
}


private void myToolbarButton_Click
 (object sender, System.EventArgs e)
{
 // Perform the plugin action here.
 this.Application.MessageBox("Toolbar button was clicked");
}
```

Modifying the Context Menu

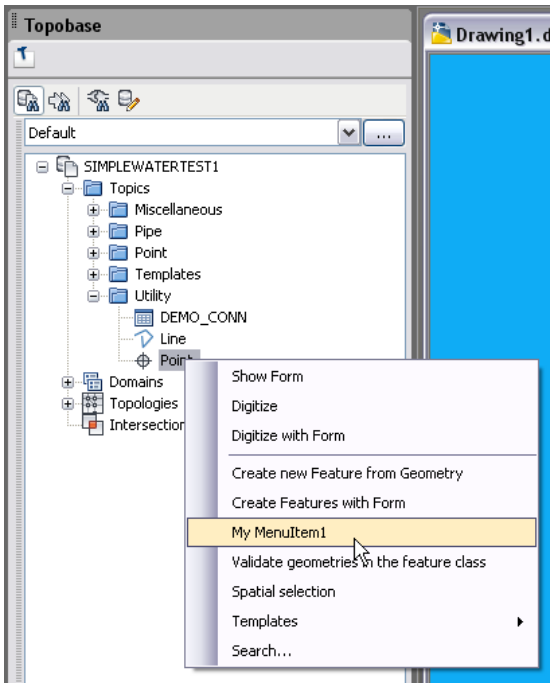
To create a new menu item within the Document Explorer's context menu:

1. Create a public class variable of type `Topobase.Forms.MenuItem`. This will serve as the communication point between Topobase and your plugin.
2. Override the `DocumentPlugIn` class's `OnInitMenus` event so you can modify the menus at the correct time after they have been created.

3. Obtain a reference to the specific document's context menu you want to modify. There are a collection of `MenuItem` menus, each assigned to a specific feature class. This way you can add menu items only to the context menu of the particular feature class you want.
4. Insert a new menu item using the `Add` method of the menu's `MenuItems` collection
5. Add an event handler for the `Click` event of the new menu item.

The following example demonstrates these steps. It adds a new menu item to the context menu of "Point" feature classes. When you right click on a "Point" feature class in the Document Explorer, you will see a new menu item called My Menu Item which will display a message box when selected.

```
public Topobase.Forms.MenuItem myMenuItem1 =
 new Topobase.Forms.MenuItem();

public override void OnInitMenus
 (object sender, Topobase.Forms.Events.MenusEventArgs e)
{
 // Obtain the context menu to which you want to add
 // your new MenuItem.
 Topobase.Forms.Menu pointMenu =
 e.Menus.Item(Topobase.Data.FeatureClassType.Point);

 // Add your MenuItem to the menu.
 pointMenu.MenuItems.Add(myMenuItem, "My Menu Item");
 myMenuItem1.Click += new Topobase.Forms.Events.
 MenuItemClickEventHandler(MyMenuItem_Click);
}


private void MyMenuItem_Click
 (object sender, Topobase.Forms.Events.MenuItemClickEventArgs e)
{
 // Perform the plugin action here.
 this.Application.MessageBox("Menu Item Clicked");
}
```



A menu item has been added to the context menu of the Point class.

Creating a Dialog Plugin

About Dialog Plugins

Dialog plugins modify the toolbars, menus, and other user-interface elements of the feature dialog boxes, and are used to provide added features while users are viewing and manipulating individual feature attributes.

Such plugins are derived from the `Topobase.Forms.DialogPlugIn` class:

```
public class MyDialog : Topobase.Forms.DialogPlugIn
{
}
```

Creating Toolbar Items

To create a toolbar icon, create a public variable of type `Topobase.Forms.ToolBarButton`. Override the `DialogPlugIn` class's `OnInitToolBar` event, obtain a reference to the dialog toolbar, and insert the new button using the `Add` method of the toolbar's `Buttons` collection. You can then create an event handler for the new toolbar button and can respond to toolbar events.

```
public Topobase.Forms.ToolBarButton myToolbarButton =
 new Topobase.Forms.ToolBarButton();

public override void OnInitToolBar
 (object sender, Topobase.Forms.Events.ToolBarEventArgs e)
{
 // Add a new toolbar button to the toolbar of the
 // generic dialogs.
 e.ToolBar.Buttons.Add
 (myToolbarButton,
 "Topobase",
 "My Toolbar Button");
 myToolbarButton3.Click +=
 new EventHandler(myToolbarButton_Click);
}


// Event is raised when the button is clicked.
private void myToolbarButton_Click
 (object sender, System.EventArgs e)
{
 // Perform the plugin action here.
 this.Application.MessageBox("Toolbar button clicked");
}
```

Creating a Menu Bar

Generic dialog boxes generally do not have menu bars, but it is possible to add them. To create a menu, create public variables of type `Topobase.Forms.MenuItem` for each menu title and menu item. Override the `DialogPlugIn` class's `OnInitMenu` event, obtain a reference to the application menu bar, and insert a new menu title using the `Add` method of the menu's `MenuItems` collection. Submenus for each menu title can be added using the

`Add` method of the menu title's `MenuItems` collection. You can then create an event handler for the new menu items and can respond to menu events.

```
// Create a Main Menu Item "File".
public Topobase.Forms.MenuItem mnuFile =
 new Topobase.Forms.MenuItem();

// Create two Menu Items as sub-menus "Open"
public Topobase.Forms.MenuItem mnuFileOpen =
 new Topobase.Forms.MenuItem();


public override void OnInitMenu
 (object sender, Topobase.Forms.Events.MenuEventArgs e)
{
 // Add the Main Menu Item.
 e.Menu.MenuItems.Add(mnuFile, "&File");

 // Add SubMenuOItems to the Menu "File"
 mnuFile.MenuItems.Add(mnuFileOpen, "&Open");

 //  Bind the event handlers.
 mnuFileOpen.Click += new Topobase.Forms.Events.
 MenuItemClickEventHandler(mnuFileOpen_Click);
}


private void mnuFileOpen_Click
 (object sender, Topobase.Forms.Events.MenuItemClickEventArgs e)
{
 // Perform the plugin action here.
 this.Application.MessageBox("File Open clicked");
}
```

Adding Controls to a Dialog

Topobase-specific controls such as buttons and list boxes can be added to dialog boxes using the Topobase Administrator (see the Sample 16 help documentation for an example). To create a plugin that responds to events from such controls, create variables of the correct `Topobase.Forms` type for each control. When the plugin is loaded, obtain references to the dialog controls using the `this.Dialog.Controls.ApiItem` method using the name

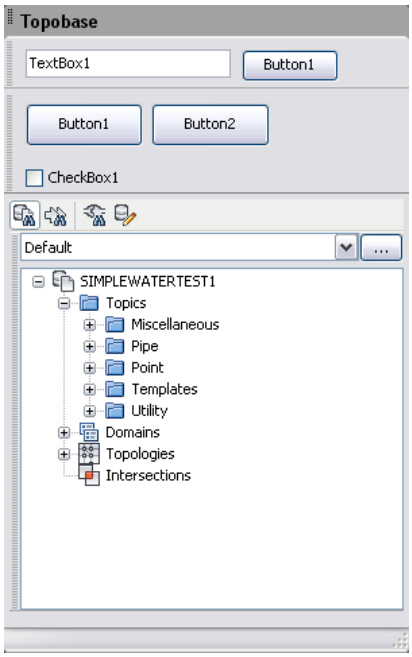
of the control assigned through Topobase Administrator. You can then create an event handler for the dialog control and can respond to control events.

```
private Topobase.Forms.Button myButton;

public MyDialog()
{
 // We cannot get access to the dialog controls in the
 // constructor. Instead, we will create an event handler
 // for the load event, and get access to the dialog controls
 // in there.
 this.Load += new EventHandler(MyDialog_Load);
}


private void MyDialog_Load(object sender, EventArgs e)
{
 // Get a reference to the button already added to the
 // dialog using Topobase Administrator.
 myButton1 = this.Dialog.Controls.ApiItem("$myButton")
 as Topobase.Forms.Button;

 // Add an event handler to catch when the button is selected.
 myButton1.Click += new System.EventHandler(MyButton_Click);
}


private void MyButton_Click
 (System.Object sender, System.EventArgs e)
{
 // Perform the plugin action here.
 this.Application.MessageBox("Button clicked");
}
```

Creating a Application Flyin

About Application Flyins

An application flyin is a dockable, moveable window that can be docked within the Topobase application part of the Topobase task pane. All docking operations are performed automatically.



Docked application flyin at the top of the Topobase task pane.

Flyins are based on either one of two differnt kinds of Topobase forms: `Topobase.Forms.Desktop.ApplicationFlyIn` and `Topobase.Forms.FlyIns.ApplicationFlyIn`. Flyins based on the `Desktop.ApplicationFlyIn` class can only be used in the desktop Topobase Client while flyins using the `FlyIns.ApplicationFlyIn` class can be used with the web-based Topobase Client as well. Desktop application fly-ins can use any Windows Form component (such as buttons, text boxes, or tree views) while the more universal style are limited to only the Topobase form components.

This sample shows the code behind a `Topobase.Forms.FlyIns.ApplicationFlyIn` fly-in which contains a single button. In response to a button click, it displays a message box.

```
public class MyApplicationFlyIn :
 Topobase.Forms.FlyIns.ApplicationFlyIn
{
 private void MyApplicationFlyIn_Load
 (object sender, System.EventArgs e)
 {
 // Set the flyin title.
 this.Text = "Sample FlyIn";
 }


 // This flyin contains a single button.
 private void Button1_Click
 (object sender, System.EventArgs e)
 {
 this.Application.MessageBox(this.TextBox1.Text);
 }
}
```

Creating a Document Flyin

About Document Flyins

A document flyin is a dockable, moveable window that can be docked within the Topobase document tabs of the Topobase task pane. All docking operations are performed automatically.

Flyins are based on either one of two different kinds of Topobase forms: `Topobase.Forms.Desktop.DocumentFlyIn` and `Topobase.Forms.FlyIns.DocumentFlyIn`. Flyins based on the `Desktop.DocumentFlyIn` class can only be used in the desktop Topobase Client while flyins using the `FlyIns.DocumentFlyIn` class can be used with the web-based Topobase Client as well. Desktop document fly-ins can use any Windows Form component (such as buttons, text boxes, or tree views) while the more universal style are limited to only the Topobase form components.

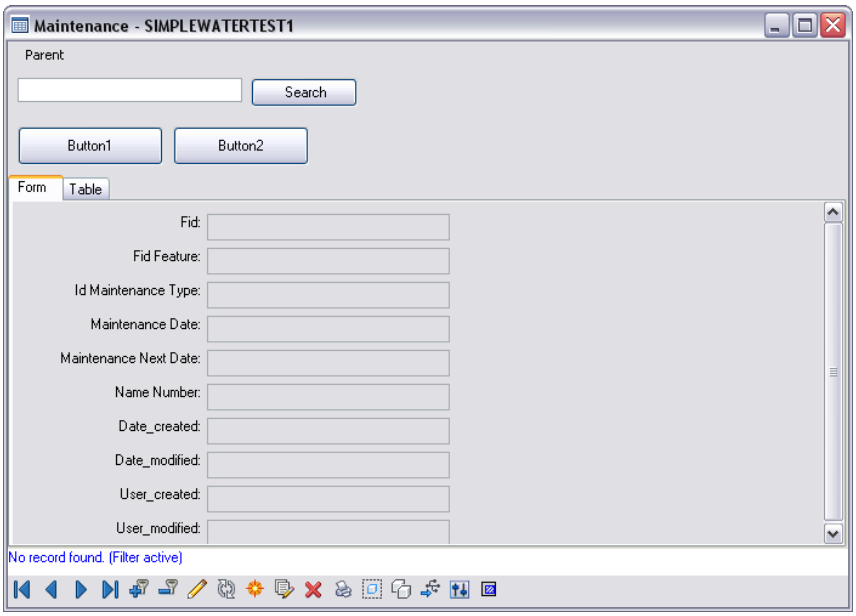
This sample shows the code behind a `Topobase.Forms.FlyIns.DocumentFlyIn` fly-in which contains a single button. In response to a button click, it displays a message box.

```
public class MyDocumentFlyIn :
 Topobase.Forms.FlyIns.DocumentFlyIn
{
 private void MyDocumentFlyIn_Load
 (object sender, System.EventArgs e)
 {
 // Set the flyin title.
 this.Text = "Sample FlyIn";
 }


 // This flyin contains a single button.
 private void Button1_Click
 (object sender, System.EventArgs e)
 {
 this.Application.MessageBox(this.TextBox1.Text);
 }
}
```

Creating a Dialog Flyin

About Dialog Flyins

A dialog flyin is a frame within Topobase dialog boxes. Unlike other kinds of flyins, these cannot be undocked and moved.



Docked dialog flyin at the top of the form showing buttons and a text box.

Flyins are based on either one of two different kinds of Topobase forms: `Topobase.Forms.Desktop.DialogFlyIn` and `Topobase.Forms.FlyIns.DialogFlyIn`. Flyins based on the `Desktop.DialogFlyIn` class can only be used in the desktop Topobase Client while flyins using the `FlyIns.DialogFlyIn` class can be used with the web-based Topobase Client as well. Desktop dialog fly-ins can use any Windows Form component (such as buttons, text boxes, or tree views) while the more universal style are limited to only the Topobase form components.

This sample shows the code behind a `Topobase.Forms.FlyIns.DialogFlyIn` fly-in which contains a single button. In response to a button click, it displays a message box.

```
public class MyDialogFlyIn : Topobase.Forms.FlyIns.DialogFlyIn
{
 private void MyDialogFlyIn_Load
 (object sender, System.EventArgs e)
 {
 // Set the flyin title.
 this.Text = "Sample FlyIn";
 }


 // This flyin contains a single button.
 private void Button1_Click
 (object sender, System.EventArgs e)
 {
 this.Application.MessageBox(this.TextBox1.Text);
 }
}
```

User Interaction

Using Forms

Basics of Using Forms

When designing plugin and flyin forms, be sure to take into account the size and shape of the area where the form will be displayed. Plugin forms and flyin forms based on `Topobase.Forms.Desktop.ApplicationFlyIn` allow you to use the standard Windows `Anchor` and `AutoSize` properties for controls, which can help you create forms that are useable and attractive in a wide variety of situation.

Using Topobase Forms in Visual Studio

See Using Topobase Forms in Visual Studio (page 130) for information on setting up Visual Studio to use Topobase form classes.

Topobase Forms Controls

A wide variety of standard and Topobase-specific controls are available for use in forms based on Topobase classes. The following table lists all the available

controls from the *Topobase.Forms.dll* library. If you need to use controls that are not in this list, you can create a standard Windows form and use any control (see Sample 115 for an example). However, the resulting plugin will only work in the desktop client and not in the web client.

Control	Samples demonstrating the control
Button	16, 17, 20, 22, 28, 29, 31, 34, 41, 45, 46, 50, 53, 56, 59, 61, 62, 67, 68, 70, 71, 74, 76, 77, 79, 80, 100, 102, 111
Canvas	76, 80
CheckBox	62, 68, 76, 78 ,79, 111
CheckedListBox	71, 76
ComboBox	53, 62, 76
DateTimePicker	50, 62, 76
DirectoryTextBox	51, 62, 76
FileTextBox	51, 62, 76
FileUploadBox	34, 62, 76
Grid	41
GroupBox	62, 76
Label	20, 28, 34, 41, 51, 53, 62, 76, 78, 100
Line	76
ListBox	16, 20, 28, 29, 30, 31, 53, 56, 62, 71, 74, 76
MapButton	30, 62, 76
Matrix	45, 62, 76
MenuControl	76, 77
Panel	62, 76

Control	Samples demonstrating the control
PictureBox	20, 62, 76
PictureComboBox	102
RadioButton	62, 76
Splitter	none
TabControl	62, 79
TextBox	20, 22, 28, 31, 46, 47, 59, 62, 67, 68, 70, 76, 77, 78, 79, 100
ToolBarControl	44, 62, 76
TreeView	29, 48, 62, 76
WebBrowser	78

Progress Bar

Progress bars dialog boxes are used to display the current progress of a task which will take a long time to complete. To display a progress bar dialog box, first create a `Topobase.Data.Util.StatusForm<>` object. The constructor requires a reference to a `BackgroundJob<>` delegate. The progress bar is displayed by calling its `ShowDialog()` method. This triggers the background job, which is where the ongoing task is performed. During the processing of the task, regular updates to the `Value` property of the `IStatusDisplay` parameter will inform the user of the current task status. Once the background job is complete, the progress bar is automatically removed.

```
// This method does something long and complicated with a
// topobase connection. It shows a progress bar while doing so.
public void DoComplicatedWorkAndShowProgress(TBConnection connec
tion)
{
 // Define which method will do the complicated work
 BackgroundJob<TBConnection> job =
 new BackgroundJob<TBConnection>(Work);

 // Create the status form
 using (StatusForm<TBConnection> form =
 new StatusForm<TBConnection>(job))
 {
 // Set some example values.
 form.CancelButtonEnabled = true;
 form.Type = TopobaseProgressBarType.ProgressBarWithPercent;

 // Set the parameter for the method.
 // You can omit this if your method does not need
 // any parameters.
 form.SetParameters(connection);

 // Show the status form - this will call the method
 form.ShowDialog();
 }
}


// The status form will call this method in a background thread.
// A background thread is used to keep the user interface
// responding. If you use a foreground thread, the user interface

// does not react when the user wants to move or resize a window.
private void Work
 (IStatusDisplay status, params TBConnection[] connections)
{
 // Get the parameter that have been set with SetParameters.
 TBConnection connection = connections[0];

 // Set the values of the status form.
 status.Minimum = 0;
 status.Maximum = 42;
 status.Value = 0;
```

```
  status.Status = "Doing important work";

  // Do the actual work
  for (int i = 0; i < 42; i++)
  {
  // We would do something complicated here 42 times

  // Update the progress bar
  status.Value += 1;

  // Abort if the user presses cancel.
  // The user can press cancel because we enabled the cancel
  // button in the calling code.
  if (status.CancelRequested)
  {
  return;
  }
  }
 }
```

Message Box and Input Box

The `Topobase.Forms` namespace includes many simple dialog boxes to give information to or take input from the user. Unlike the Windows Form equivalents, these are compatible with Topobase Web Client.

Message Box

`Topobase.Forms` contains a simple message box than can display a message and an optional title.

```
Topobase.Forms.Application.MessageBox("Message", "Title");
```

Input Box

An input box can be used to get a single line of text from the user. It can be displayed asynchronously, so your plugin can continue processing while the user types.

```
// Declare the input box variable.
public Topobase.Forms.Interaction.InputBox myInputBox =
 new Topobase.Forms.Interaction.InputBox();

public void Init()
{
 // Assign the input box event handler.
 myInputBox.Executed += new Topobase.Forms.Events.
 InputBoxEventHandler(MyInputBox_Executed);

 // Show the input box.
 myInputBox.ShowAsynchronous("What\'s your Name ?", "Name");
}


private void MyInputBox_Executed
 (object sender, Topobase.Forms.Events.InputBoxEventArgs e)
{
 Application.MessageBox("Your Name is:" + e.Value);
}
```

Another kind of input box has multiple text boxes, allowing more complicated input while retaining a simple programming interface.

```
// Declare the input box variable.
public Topobase.Forms.Interaction.MultiInputBox myMultiInputBox =

 new Topobase.Forms.Interaction.MultiInputBox();


public void Init()
{
 // Assign the input box event handler.
 myMultiInputBox.Executed += new Topobase.Forms.Events.
 CancelEventHandler(MyMultiInputBox_Executed);

 // Show the input box.
 myMultiInputBox.Add("p1", "Parameter 1");
 myMultiInputBox.Add("p2", "Parameter 2", "Test");
 myMultiInputBox.Add("p3", "Parameter 3");
 myMultiInputBox.Show();
}


private void MyMultiInputBox_Executed
 (object sender, Topobase.Forms.Events.CancelEventArgs e)
{
 if (e.Cancel)
 {
 Application.MessageBox("Canceled");
 }
 else
 {
 string outString = myMultiInputBox.Item("p1");
 outString = outString + "," + myMultiInputBox.Item("p2");
 outString = outString + "," + myMultiInputBox.Item("p3");
 Application.MessageBox("Result:" + outString);
 }
}
```

Installation

The *.tbp* file for libraries that expose application plugins should have a line like the following for each plugin:

```
<ApplicationPlugIn ClassName="MyApplicationPlugin" />
```

The *.tbp* file for libraries that expose document plugins should have a line like the following for each plugin:

```
<DocumentPlugIn ClassName="MyDocumentPlugIn" />
```

The *.tbp* file for libraries that expose dialog plugins should have a line like the following for each dialog plugin:

```
<DialogPlugIn ClassName="MyDialogPlugIn" />
```

The *.tbp* file for libraries that expose application flyins should have a line like the following for each flyin:

```
<ApplicationFlyIn ClassName="MyApplicationFlyIn"/>
```

The *.tbp* file for libraries that expose document flyins should have a line like the following for each flyin:

```
<DocumentFlyIn ClassName="MyDocumentFlyIn2" />
```

The *.tbp* file for libraries that expose dialog flyins should have a line like the following for each flyin:

```
<DialogFlyIn ClassName="MyDialogFlyIn" Name="" />
```

Plugins and Visual Studio

Installing the Project Template

Autodesk Topobase includes a generic project template for creating plugins in Visual Studio 2005. To install it, copy the project template named *CSSimpleTopobasePlugIn.zip* from the *<Topobase install directory>\Development\VS Templates\ProjectTemplates\Topobase* directory to the Visual Studio custom template directory (usually *My Documents\Visual Studio 2005\Templates\ProjectTemplates\C#*).

To create a stand-alone plugin project using the template:

1. Start Visual Studio.
2. Select the File ➤ New ➤ Project menu item.
3. In the New Project dialog box, select the Visual C# Project Type.
4. Select SimpleTopobasePlugIn from the list of templates and create a project.

5 Once the project has been created, check the project references to make sure that the references to the Topobase libraries are valid.

Installing the Item Templates

Plugins can use regular Windows forms and user-controls to interact with the user. However, any plugin that uses these will only work in the desktop environment and will not work in the Topobase web clients. To create user interface elements that are usable in both environments, you can use form templates provided by Topobase in your projects. These forms are limited to the controls provided in the *Topobase.Form.dll* library.

To install the templates, copy the item template files (which are all *.zip* compressed files) from the *<Topobase install directory>\Development\VS Templates\ItemTemplates\Topobase* directory to the Visual Studio custom template directory (usually *My Documents\Visual Studio 2005\Templates\ItemTemplates\Visual C#* and *My Documents\Visual Studio 2005\Templates\ItemTemplates\Visual Basic*).

To add Topobase forms to your project:

1 Select the Project ➤ Add New Item menu item.

2 Select the appropriate plugin component type from the available templates in the My Templates section of the template list.

Installing User Controls

Autodesk Topobase includes a number of controls for use in Autodesk Topobase forms, some of which replicate standard Windows user control and some of which are unique to Autodesk Topobase. To add Autodesk Topobase controls to the Visual Studio Toolbox:

1 Right-click in the Toolbox, and select the Choose Items... menu item.

2 Select Browse in the Chose Toolbox Items dialog box.

3 Select *Topobase.Forms.dll* from the Autodesk Topobase Client */bin/* directory.

Creating Survey File Format Plugins

Creating a Survey File Format Module

The Topobase Survey wizard allows reading of survey field measurements. It supports the CPlan RO and Leica GSI data formats by default, but it is possible to read other formats by creating your own plugins that translate the data in a file to data structures Topobase Survey understands. Survey file format plugins are based on the `FileFormatPlugIn` class.

```
// Create a plugin for reading survey measurements
// from a custom file format.
class SampleImportPlugin : FileFormatPlugIn
{
    // Be sure to call the base class constructor in your own
    // constructor.
    public SampleImportPlugin(TBConnection connection)
        : base(connection)
    {
    }
}
```

Survey file format plugins need to override the following abstract properties and methods:

- **Author** - Read-only property returning a string containing the organization name. The return value can be null.
- **FormatName** - Read-only property returning a string containing the name of the format. This string will be used in the File Format combo box of the Import Measurements page of the Topobase Survey wizard. The return value cannot be null.
- **StartImport()** - Method that performs the file reading operation. Each line of text from the source file is retrieved using the base class's `ReadLine()` method. The information is then returned to Topobase using the base class's `InsertCoordinate()`, `InsertMeasurement()`, and `InsertStation()` methods. See Implementing StartImport for more information.

Implementing StartImport

Reading the Survey File

The survey file is selected by the user in the Topobase Survey wizard. The file is read one line at a time using the base class's `ReadLine()` method. Usually each line of text corresponds to one survey measurement, although this is not required. `ReadLine()` returns a string containing one line from the file, and automatically advances which line is returned each time it is called. When there are no more lines in the file, `ReadLine()` returns null.

```
// Read the first line of the file.
string line = this.ReadLine();
while (line != null)
{
    // Parse the line and inform Topobase what it indicates

    // Read the next line of the file.
    line = this.ReadLine();
}
```

Determining Instrument Type

Just like the file, the type of instrument is selected by the user in the Topobase Survey wizard. The type of instrument and settings for the instrument determine the units, error correction, and other meanings of the values read from the file. It is important to check the type of instrument selected by the user to make sure the measurements in the file are meaningful. The base class property `SelectedInstrumentType` contains one of an enumerated set of values describing the instrument type.

SelectedInstrumentType value	Meaning
Digitizer	Direct measurements of northing and easting at each point.
Nivellement	Measurements taken from a dumpy level or other form of builder's level.
Tachymeter	Measurement of direction and distance from station points, measured by indirect methods (such as optically or by laser).

SelectedInstrumentType value	Meaning
Tape	Measurement of direction and distance from station points, measured by physical methods (such as by tape or chain).
Transformation	Direct measurement of northing and easting at each point, such as by a GPS device.

Suppose a line was read from the file containing a measurement that is only valid for tape or tachymeter survey measuring techniques. We can check that the instrument selected from the user is valid, and perform different actions depending on which instrument was selected:

```
switch (this.SelectedInstrumentType)
{
    case InstrumentType.Tachymeter:
        // Interpret the data as coming from a tachymeter.
        break;
    case InstrumentType.Tape:
        // Interpret the data as coming from tape measurments.
        break;
    case InstrumentType.Transformation:
        throw new DemoImportException
          ("Invalid line for GPS\Transformation instrument");
        break;
    default:
       throw new DemoImportException("Unexpected instrument selec
ted");
}
```

Interpreting Survey Measurements From the File

When reading a file containing survey measurements, you will find two basic kinds of data: stations and measurements based off a particular station. When you have parsed the information for a station, call the base class's `InsertStation()` method. At a minimum, this method requires a unique identifier for the station, a field code, and the height of the instrument. This method returns an integer FID for the station

Interpreting GPS Coordinates From the File

When reading a file containing GPS coordinates or other measuring system where each point is measured directly, you should see only one kind of

information in the file - the point coordinate. When you have parsed the information for one point, call the base class's `InsertCoordinate()` method. At a minimum, this method requires a unique identifier for the point, a field code, and the easting, northing, and elevation of the point. The method also requires the FID of a station which is used to identify the session for all the points you are inserting. To get this fid, call the `InsertStation()` method with no parameters and store the integer return value. Use that integer for all you calls to `InsertCoordinate()`.

```
public override string StartImport()
{
    string identifier;
    string fieldCode;
    double easting;
    double northing;
    double altitude;

    // Get a blank stationFID
    long stationFID = this.InsertStation();
    string line = null;

    // Read the file one line at a time until there are no
    // more lines.
    while ((line = this.ReadLine()) != null)
    {
        //
        // NOT SHOWN - parse the line of text to extract the
        // identifier, fieldCode, easting, northing, and
        // altitude.
        //

        this.InsertCoordinate(
           stationFID,
           identifier,
           fieldCode,
           PointType.Base,
           easting,
           northing,
           altitude);
    }
}
```

Handling Errors

If an error occurs during the reading of the measurement file, you can report the nature of the error to the user by using the base class's `AddErrorMessage()` and `ThrowImportException()` methods. `AddErrorMessage()` take a single string parameter which describes the error (such as a value being out of bounds). `ThrowImportException()` will display an error message to the user containing all the text from all `AddErrorMessage()` calls made up to that point.

```
catch (DemoImportException e)
{
   // Handle import exception
   this.AddErrorMessage(e.Message);
   this.ThrowImportException();
}
```

Creating Option Pages

Creating an Option Page Module

An option page is a form within the Application Options or Document Options dialog box. A plugin can be used to add new forms to these dialogs. These plugins are objects based on the `Topobase.Forms.ApplicationOptionPage` class or `Topobase.Forms.DocumentOptionPage` class.

```
// Create an option page for the Application
// Options dialog box.
public class MyOptionPage :
  Topobase.Forms.Desktop.ApplicationOptionPage
{
}
```

Option pages are based on either Windows user controls or Topobase Forms. Option pages using Windows user controls can only be used in the Topobase Desktop Client while option pages using Topobase Forms can be used with both the Topobase Desktop Client and the Topobase Web Client.

Reading and Writing User Settings

Topobase includes a built-in mechanism for handling user settings. Settings are stored in the TB_SETTINGS table of the TBSYS User database, which can

be accessed using methods of the `Topobase.Settings` namespace. Application option pages access these methods through the `this.Application.Settings` property and document option pages access them though the `this.Document.Settings` property, but their use is the same for both. Be sure your project references the *Topobase.Settings.dll* library to access these methods.

The TB_SETTINGS table contains the following columns:

Table Column	Type
ID	number(10)
User_ID	number(10)
ItemThema	varchar2(255)
ItemKey	varchar2(255)
ItemValue	varchar2(2000)

The *ItemThema* column is used to group keys together. If your application Option Page creates new setting keys, give them all the same thema name based on the name of your module.

Items in the TB_SETTINGS table can be access through the `Topobase.Settings.Database` methods `get_Item` and `set_Item` and their type-safe variants, `get_BoolItem` and `set_BoolItem` for Boolean values, `get_DblItem` and `set_DblItem` for floating point numbers, and `get_LngItem` and `set_LngItem` for integers. Using the type-safe versions is highly recommended because it can prevent conversion errors.

Settings can be assigned to all users or to specific users depending on the `Dependency` parameter of the `get_Item` and `set_Item` methods. If a setting is assigned to a specific user, the setting is also copied to the settings table for all users. This serves as a default value if you attempt to read specific user settings that have not yet been set.

The following sample demonstrates reading and writing user settings. It is based on a Topobase Form with two textbox controls. When the form is first displayed, it will access the existing settings and display them in the textboxes. When the user closes the option page dialog box, the settings will be saved.

```
// Constructor
public MyOptionPage()
{
 InitializeComponent();

 // Add event handlers for when the form is loaded
 // and unloaded.
 this.Unload += new Topobase.Forms.Events.
 UnloadEventHandler(this.MyOptionPage_Unload);
 this.Load += new System.EventHandler(this.MyOptionPage_Load);
}

private void MyOptionPage_Load(object sender, System.EventArgs e)
{
 // Set the title of your TabPage.
 this.Text = "MyOptions";
}


// This event is fired if the user selects the TabPage the first
// time.  If your data took a very long time to initialize,
// initialize it here next time instead of at the Load Event.
// This will prevent the whole Option Dialog from opening
// very slowly.
private void MyOptionPage_ActivatedFirstTime
 (object sender, System.EventArgs e)
{
 // Read the option from the settings.
 this.TextBox1.Text = this.Application.Settings.get_Item
 ("CSSampleOptions",
 "Name1",
 Topobase.Settings.Dependency.CurrentUser,
 "Test1");
}


// This is called every time the user clicks the Main TreeNode
// on the Left Tree of the Option Page.
// You can use this to reset your data if
// you have more than one Option Page Plugin,
// and the Plugins depend on one another.
private void MyOptionPage_Reset(object sender, System.EventArgs
e)
```

```
{
 // do nothing
}


private void MyOptionPage_Unload
 (object sender, Topobase.Forms.Events.CancelEventArgs e)
{
 // If the user cancels the dialog, don't save the changes.
 if (e.Cancel)
 {
 return;
 }

 // Save the Options. Note how the thema name (the first
 // parameter) is set to the name of the module to identify
 // this and any additional settings as originating from this
 // program.
 this.Application.Settings.set_Item
 ("CSSampleOptions",
 "Name1",
 Topobase.Settings.Dependency.CurrentUser,
 this.TextBox1.Text);
}
```

Installation

The *.tbp* file for libraries that expose application option pages should have a line like the following for each option page:

```
<ApplicationOptionPage ClassName="MyOptionPage" />
```

A document option page *.tbp* file should have a line like the following for each option page:

```
<DocumentOptionPage ClassName="DocumentOptionPage1"/>
```

Creating Administrator Pages

Creating an Administrator Page Plugin

Vertical application modules may also require custom types of initialization in the Topobase Administrator application. You can add new pages to the "Setup" branch of the tree by creating a plugin based on the `ApplicationAdminPage` class and you can add pages to the document branches (which are visible only after loading a workspace) by creating a plugin based on the `DocumentAdminPage` class. Sample 122 includes a plugin based on `DocumentAdminPage` - there are no samples using `ApplicationAdminPage`, but the two classes are very similar.

```
public partial class MyAdminPage :
   Topobase.Forms.Desktop.DocumentAdminPage
{
}
```

The name of the page must be set when the plugin is loaded. If you are creating a `DocumentAdminPage`-based plugin, the name will be displayed in each document tree once a workspace is loaded. If you are creating an `ApplicationAdminPage`-based plugin, the name will be displayed in the "Setup" section of the tree view. If the name property has not been set, the title of your page will be three question marks ("???").

```
// Constructor
public MyAdminPage()
{
   // Set up the event handler for the Load event so that the
   // function MyAdminPage_Load is called when the plugin
   // is first loaded.
   this.Load += new System.EventHandler(this.MyAdminPage_Load);
}


private void MyAdminPage_Load(object sender, EventArgs e)
{
   // Set the text shown in the Topobase Administrator tree view.
   this.Text = "My Custom Admin Page";
}
```

Responding to Administrator Events

Load

If you are creating a `DocumentAdminPage`-based plugin, the `Load` event is fired when the user loads a workspace in the Topobase Administrator application. If you are creating an `ApplicationAdminPage`-based plugin, this event is fired when the user runs the Topobase Administrator application. This is the preferred location for the initialization of your plugin, including setting the name of the page.

ActivatedFirstTime

This event is fired the first time the user selects the page title from the tree view on the left side of the Topobase Administrator, just before the user interface is displayed. You can perform additional initialization in this event. `ActivatedFirstTime` will be fired before the `Reset` event.

Reset

This event is fired each time the user selects the page title from the tree view. It is possible that changes made in other pages in the Topobase Administrator will make the data displayed in your form obsolete, so you can use this event to reload your data, if needed.

Close and Unload

If you are creating a `DocumentAdminPage`-based plugin, these events are fired when the user either closes the currently open workspace or exits the Topobase Administrator application with a workspace loaded. If you are creating an `ApplicationAdminPage`-based plugin, these events are fired when the user exits the TopobaseAdministrator. `Close` is fired before `Unload`. Both of these events are passed a `CancelEventArgs` parameter, and you can prevent the workspace or application from closing by setting the parameter's `Cancel` property to `True`.

Installing Administrator Plugins

The *.tbp* file for libraries that expose application administrator pages should have a line like the following for each option page:

```
<ApplicationAdminPage ClassName="MyAppAdminPage" ExecutionTarget
Desktop="True" ExecutionTargetWeb="False"/>
```

A document administrator page *.tbp* file should have a line like the following for each option page:

```
<DocumentAdminPage ClassName="MyAdminPage" ExecutionTarget
Desktop="True" ExecutionTargetWeb="False"/>
```

The *.tbp* file and the plugin *.dll* file should both be moved to the Topobase Administrator *bin* directory. If your plugin .dll has both administrator pages and other kinds of plugins, copy the *.tbp* and *.dll* files to the Topobase Client *bin* directory as well - the correct plugins will be automatically loaded in each case. To remove the plugin, delete the *.tbp* file.

Installation

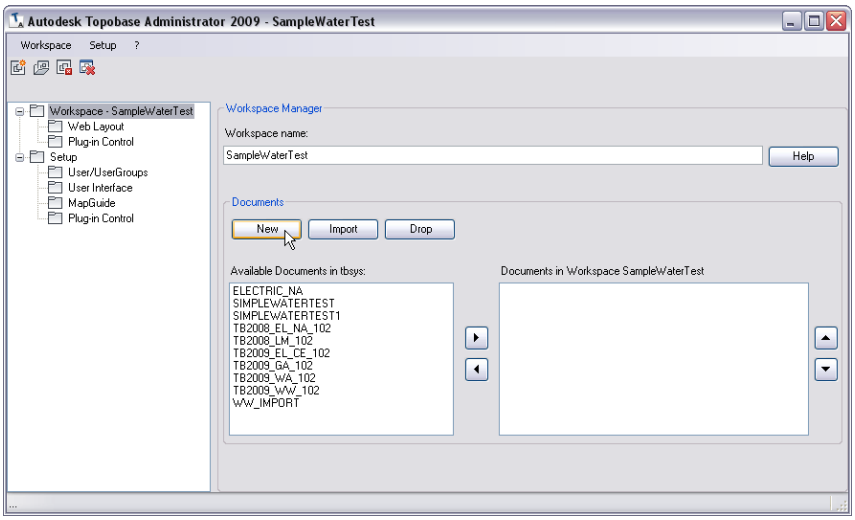
Building and Installing an Application Module

Vertical application modules are created from .NET “class library” projects. The library you get from building the project (a file with a *.dll* extension) should be copied into both the Topobase Client *bin* directory. If the dll contains administrative methods (like structure update model changing code) in addition to user code (like workflows), it will also need to be placed in the Topobase Administrator *bin* directory. The associated *.tbp* file need to be copied into those directories as well. When Topobase is started, it searches the *bin* folder for *.tbp* files and automatically loads all modules listed in those files.

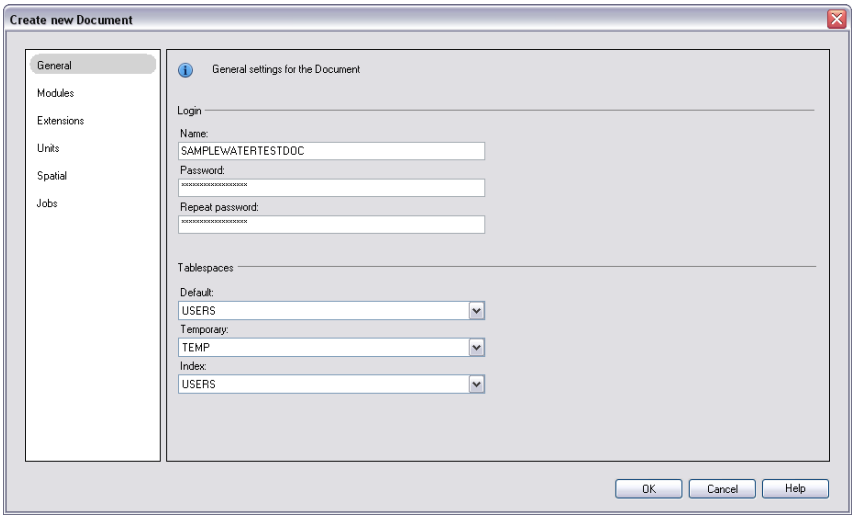
Creating a Document Using the Module

After a vertical application module has been installed, perform the following actions to create a new document:

1. Launch the Topobase Administrator application.
2. Load an existing workspace or create a new workspace (see Create a Workspace (page 152) for more information).
3. Select the New button in the Document box to create a new document.

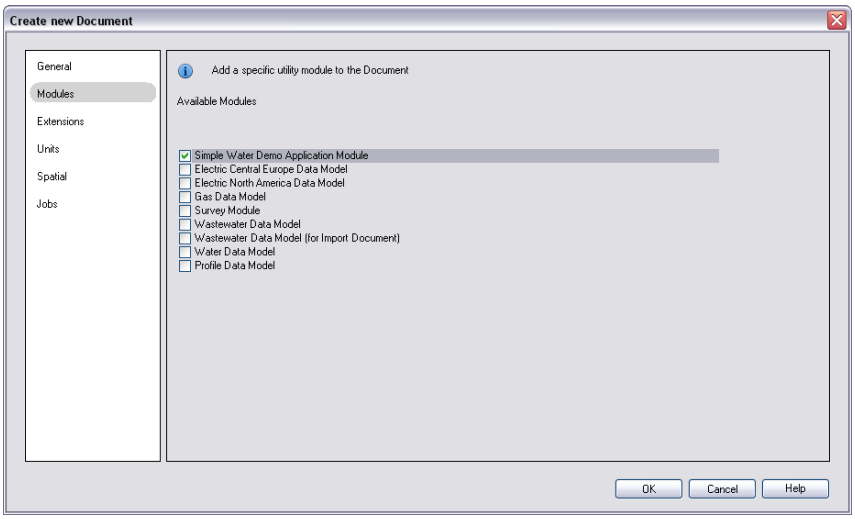


4 Enter a valid password in the Document tab of the Create New Document dialog box.



5 Select the Module tab on the left side of the Create New Document dialog box.

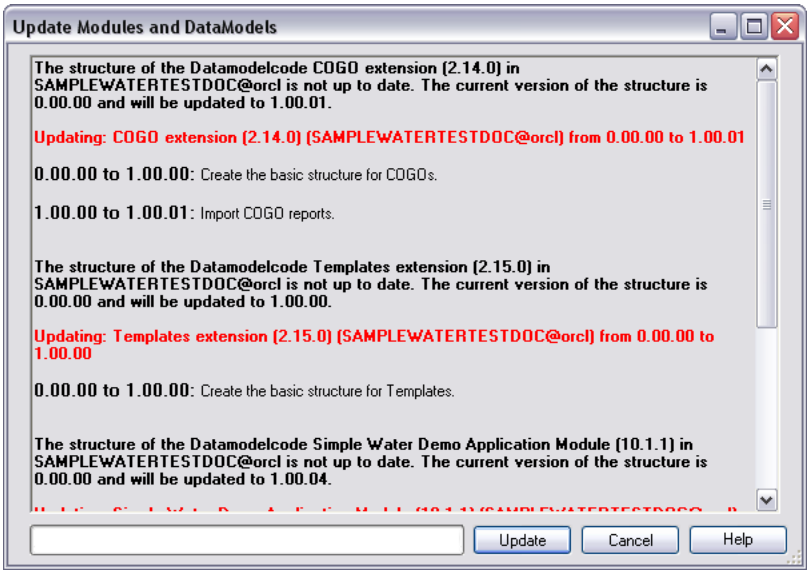
6 Check the name of the module (for example, “Simple Water Demo Application Module”) from the list of available modules.



NOTE If your application module is not in the list, it means the *.dll* and *.tbp* files are not in the Topobase Administrator *\bin* directory. Close the Administrator, copy the files from the Topobase Client *\bin* directory, restart the Topobase Administrator, and try again.

7 Press OK. This will begin the process of creating a new document.

8 After a series of status message boxes, you will see the Update Modules and DataModels dialog box listing all the updates specified in the structure update plugin of the application module. Press Update.



9 This will create the data model and apply all the updates specified in the structure update plugin. When the process is complete, press Close.

10 The document based on the vertical application module has now been created in the workspace. Select the Data Model and Workflows tree elements to see the data model and workflows provided by the module.

Manually Installing a Stand-Alone Workflow

Workflows are created from .NET "class library" projects. The library you get from building the project (a file with a *.dll* extension) should be copied into the Topobase Client *bin* directory. An associated *.tbp* file need to be copied into those directories as well.

Workflows need to be associated with particular workspace using the Autodesk Topobase Administrator, and are loaded when the workspace is loaded in the Client. Use the following steps to associate a workflow with a workspace:

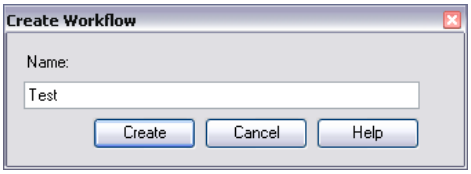
1 Start Autodesk Topobase Administrator.

2 Use the Workspace ➤ Open... menu to open the workspace.

3 Select the folder labeled Workflows in the left pane.

4 In the Workflow Administrator area, click Create.

5 In the Create Workflow dialog box enter a name for the workspace and click Create.



6 In the Workflow Administrator area, type the description of the workflow and select a bitmap or icon file containing a 16 by 16 pixel image to represent the workflow in the client application. Select which client applications the workflow is available

7 Add code into the Script Code edit field which calls the plugin when the user activates the workflow. The code consists of a single call to `RunMethod` within the `Run` subroutine. The first parameter is the filename of the workflow library, the second parameter is the namespace and class name of the workflow, and the third parameter is the name of the entry point method:

```
Sub Run
 Me.RunMethod("Sample.dll", "Sample.MyWorkflowPlugIn", "MyWork
flowEntrypoint")
End Sub
```

When you are done, click Validate to make sure the code is correct.

8 Click Save. The workflow has now been added to the workspace document and will be listed in the document's Workflow Explorer in the appropriate client application.

Installing a Stand-Alone Plugin

Plugins and flyins are created from .NET "class library" projects. The library you get from building the project (a file with a *.dll* extension) should be copied into the Topobase Client *bin* directory. An associated *.tbp* file need to be copied into those directories as well. When Topobase is started, it searches the *bin* folder for *.tbp* files and automatically loads the plugin and flyin libraries listed in those files. No further steps are required.

Developer Samples 6

Introduction

Autodesk® Topobase™ provides a large number of samples showing how you can use the API. These are provided in both Visual Basic and C# formats. They cover a broad range of categories, such as creating users, connecting to the database, customizing the user interface, and manipulating features.

This chapter describes how to build, install, and run the samples, and gives a brief description of what they do. For more details, see the sample code itself. The sample code is installed into *<topobase>\Development\Samples* where *<topobase>* is the Autodesk Topobase installation directory, typically *C:\Program Files\Autodesk Topobase Client 2009*.

NOTE The samples are numbered, but since obsolete samples have been removed the numbers are not always consecutive. There are over 70 samples in each of the C# and VB.Net formats.

Most of the samples use a Topobase plugin configuration file (*.tbp*). For more information, see TBP File Format (page 285).

IMPORTANT Most of the samples can be run independently. However, some of them use the required features and workspace set up by Sample 01 so you must build and run Sample 01 first. See Sample 01 - Create Structure (page 149) for details.

Some of the samples require the use of a workspace more fully featured that the one setup by Sample 01. The workspace referred to here is the demo Land Management workspace (TB2009_LM_102) whose creation is described in *Creating a Workspace* in the Client configuration chapter of the *Autodesk Topobase Installation and Configuration Guide*.

Unless otherwise stated, all the samples can be run in the desktop client or web client.

Building The Samples

When configured appropriately (see Configuring the Project to Write the Build Outputs to the Topobase Bin Folder (page 54)), during each sample build the required output files are copied to the Autodesk Topobase bin folder (that is, the required *.dll*, *.xml*, *.pdb*, *.tbp*, and/or *.exe* are copied). See the Common Steps (page 149) and Details (page 149) sections for each sample for more detailed information.

Usernames and Passwords

If you are not using the default Oracle service name `orcl`, or Autodesk Topobase system username `TBSYS` and password `TBSYS`, you will need to change them in all lines like the following:

```
Topobase.Data.TBConnection myConnection = new Topobase.Data.TBCon
nection("TBSample", "avs", "orcl", true, "TBSYS", "TBSYS", "orcl");
```

TIP The user with username `TBSample` and password `avs` is created by Sample 01 and is used in several other samples.

Topobase Database Server (TBSYS)

The Autodesk Topobase database server `TBSYS` can be installed on the same computer as the Autodesk Topobase Client version or on a separate database server. Use the Autodesk Topobase Server Administrator to install the Autodesk Topobase server.

For more information about installing TBSYS, see *Quick Start to Topobase Client Installation* in *Autodesk Topobase Installation and Configuration Guide*.

Running The Samples

Sample 01 - Create Structure (page 149) needs to be built and run *first* to create the TBSAMPLE workspace (see Create a Workspace (page 152).)

Create a Drawing Template (page 155) is also a required procedure to create the TBSAMPLE drawing table. The steps are documented at the end of Sample 01.

Common Steps

There are some steps that are required for running all of the samples:

1. Build the sample. The build process will automatically place the program or plugin into the *bin* folder. If you move the project from the Topobase *Development\Samples* folder, be sure to change the project output path so that the executable is placed in the correct location (the output path is adjusted in the Build tab of the project properties in Visual Studio). For projects that create a plugin, the associated *.tbp* file is automatically copied to the Topobase Client *bin* folder. For more information about *.tbp* files, see TBP File Format (page 285)
2. Run the sample. For applications, run the executable in the Autodesk Topobase *bin* folder. For plugins, start Autodesk Topobase, connect to the database, and open a workspace. This triggers the loading of the plugins.

To disable a plugin, remove the sample's *.dll* and *.tbp* files from the *bin* folder and close the current workspace. Remove any *.xml* and *.pdb* files associated with the project as well.

NOTE Unless otherwise noted, menu references (for example, click Workspace ➤ Create) refer to the menu in the Topobase task pane.

Details

Sample 01 - Create Structure

Purpose

This sample shows how to:

- Create a new database (Oracle) user

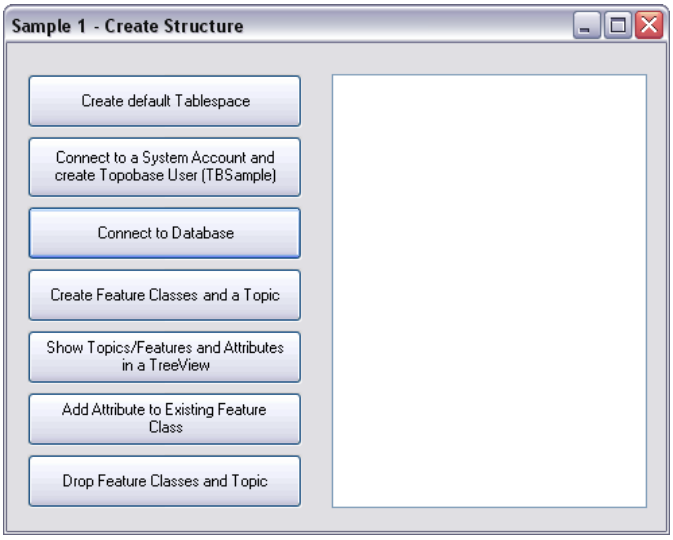
- Connect to a database
- Create feature classes
- Get information about feature classes
- Delete feature classes

The workspace and feature classes created here are also used in some of the other samples.

Procedure

To use this sample:

1. Build the sample. The plugin (*VBSample01.dll* or *CSSample01.dll*) will be placed in the Topobase Client *bin* directory. The *.tbp* file will also be copied to the Topobase Client *bin* directory.
2. Start Topobase.
3. Click the application toolbar button with the tooltip Sample 01 - Create Structure. The sample's dialog box will be displayed.



4 Click Connect to a System Account and Create a Topobase User (TBSample). This creates an Oracle user with username `TBSample` and password `avs`. This may take a few minutes to complete.

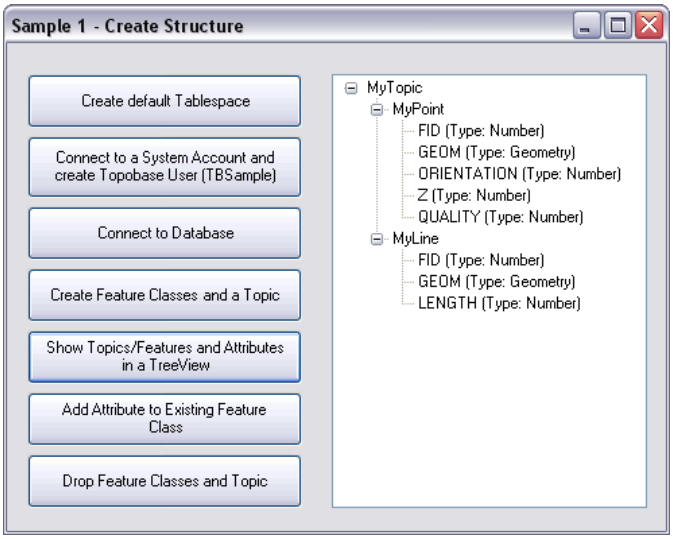
NOTE You will only need to do this step once. Subsequent attempts will display a message saying the user was already created.

5 Click Connect to Database. This connects to the Oracle database.

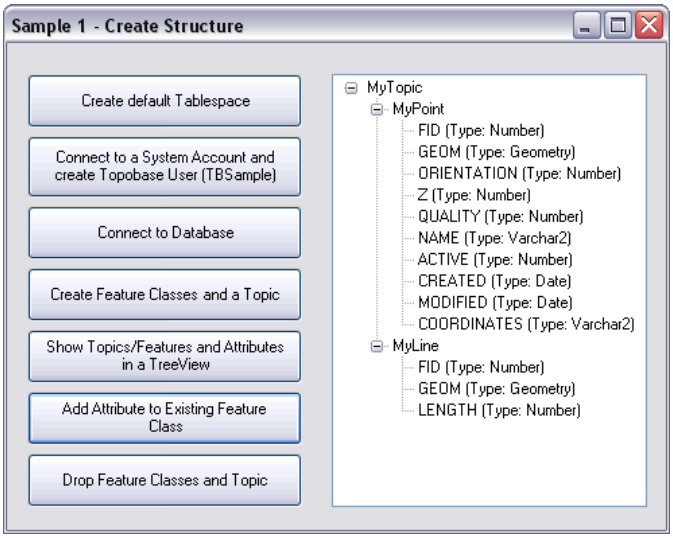
6 Click Create Feature Classes and a Topic. This creates a topic called MyTopic and two feature classes, MyLine and MyPoint.

NOTE You should only need to use this step once to create the Feature Classes and Topic. If needed, you can click Drop Feature Classes and Topic then recreate them at another time.

7 Click Show Topics / Features and Attributes in a Treeview. This shows information about the features on the right.



8 Click Add Attribute to Existing Feature Class. This adds some more attributes to the feature class.



NOTE Some of the other samples use the MyLine and MyPoint feature classes, so if you click Drop Feature Classes and Topic you should click Create Feature Classes and a Topic again before you exit the sample.

9 Exit the sample.

Create a Workspace

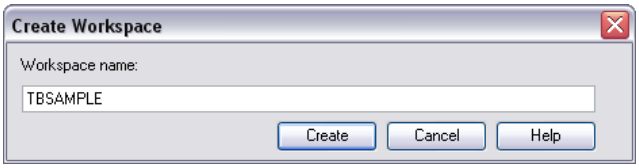
You will use the TBSAMPLE workspace in many of the following samples. It is important to create it now for later use.

To create the TBSAMPLE workspace:

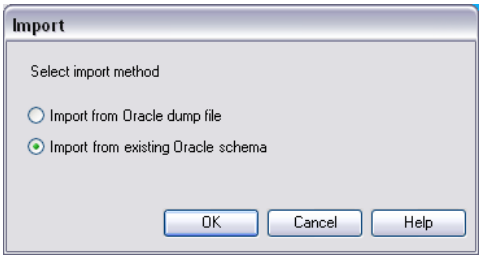
1 Start Topobase Administrator.

2 Click Workspace ➤ Create.

3 Enter TBSAMPLE for the workspace name and click Create.

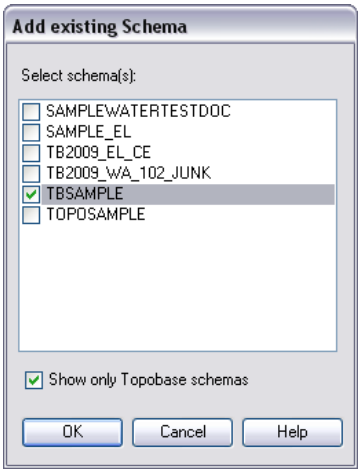


4 Click Import.

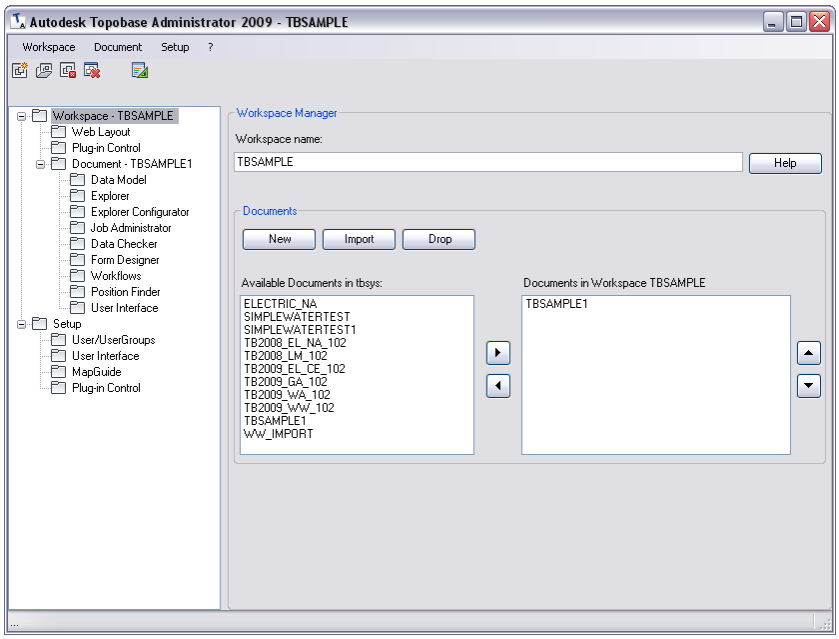


5 Click Import From Existing Oracle schema.

6 Check the TBSAMPLE schema.



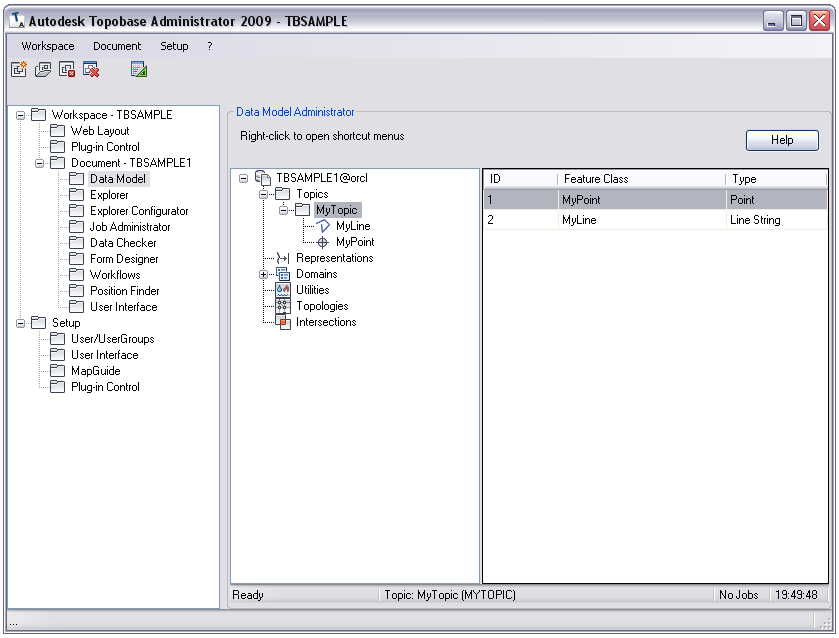
The TBSAMPLE document will now be listed in the Documents in Workspace TBSAMPLE column.



7 Set the features to be shown in the Feature Explorer:

1 Select Data Model in the left tree.

2 Check MyLine and MyPoint. These feature classes will be used in many of the following samples.



3 Click Save Tree.

8 Optionally, create a web layout for the workspace.

9 Close Topobase Administrator.

NOTE For information about creating a web layout, see *Topobase Web Configuration* in *Autodesk Topobase Installation and Configuration Guide*.

The workspace is now ready to use. This workspace is used in many of the other examples.

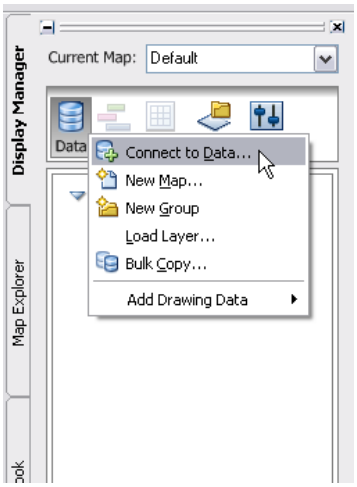
Create a Drawing Template

You will use the TBSAMPLE workspace in many of the following samples. In particular, you will generate a graphic from the points and lines in the TBSAMPLE workspace for display in the graphics pane. Before you can generate this graphic, you must create a drawing template.

For more information about drawing templates, see *Creating a Drawing Template* and other related topics in *Autodesk Topobase Installation and Configuration Guide*.

To manually create the TBSAMPLE drawing template:

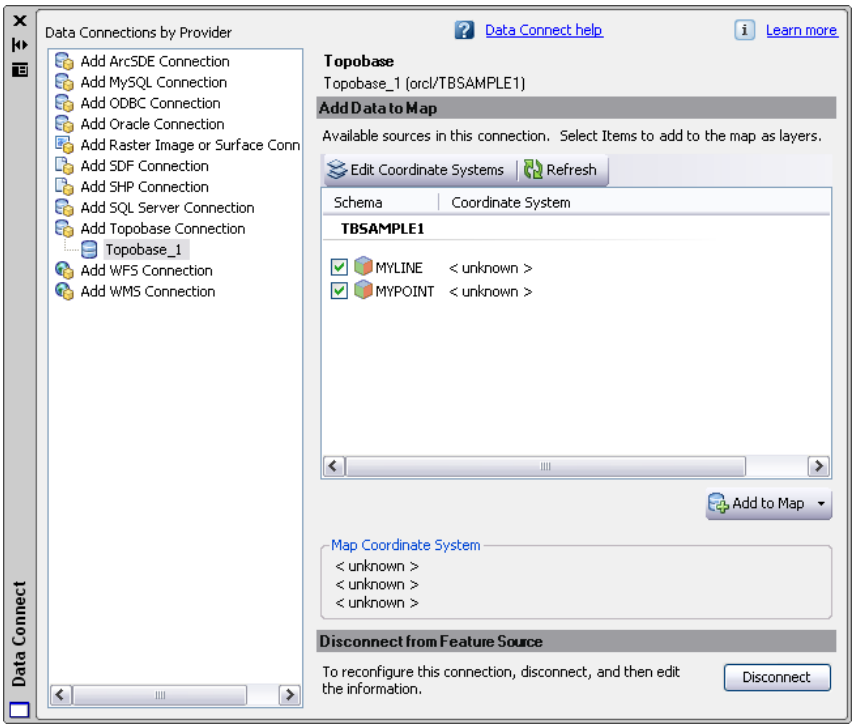
1. Start Topobase.
2. In the Open Workspace dialog box, select TBSAMPLE and click Open.
3. In the Task Pane select the Display Manager tab.
4. Click the icon labeled Data. Select Add Drawing Data... from the menu. The DATA CONNECT dialog box is displayed.



5. In the Data Connections By Provider pane click Add Topobase Connection. A form labeled Add a New Connection is displayed on the right.
6. Do not change the default value in the text box labeled Connection name. For this exercise the default value is Topobase_1.
7. In the text box labeled Service Name: type orcl.
8. In the text box labeled Topobase Main or System User name: type tbsys.
9. In the text box labeled Topobase Main or System User password: type tbsys.
10. In the dropdown box labeled Document: select TBSAMPLE.

11 Click Connect. A subnode labeled with the connection name ("Topobase_1") is displayed under the feature source node labeled Add Topobase Connection on the left. A form labeled Add Data to Map is displayed on the right.

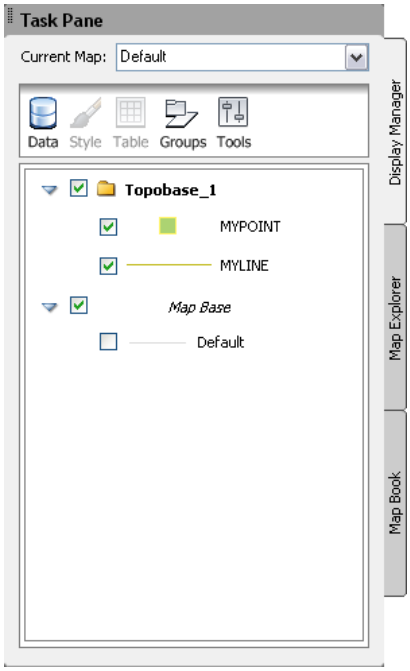
12 In the pane labeled Available sources in this connection click the checkboxes for the MYPOINT and MYLINE items. Click the Add to Map dropdown button and select Add to Map With Query. A dialog box labeled Feature Source Query is displayed.



13 In the Feature Source Query dialog list box, both TBSAMPLE:MYLINE@Topobase_1 and TBSAMPLE:MYPOINT@Topobase_1 should be listed. Click OK.

14 In the DATA CONNECT dialog box, the Data Connections by Provider pane will display two new subnodes under the Topobase_1 node. They are labeled MYLINE and MYPOINT. A form labeled Source Settings is displayed on the right. In the Display Manager tab a node labeled Topobase_1 with two subnodes labeled MYPOINT and MYLINE are

displayed. In the graphics pane the points and lines in TBSAMPLE are plotted.



15 Save the drawing template. Select Save As... from the File menu of the main menubar.

16 In the Save Drawing As dialog box select AutoCAD Drawing Template (*.dwt) from the dropdown box labeled Files Of Type. Name the file TBSAMPLE. Browse to *C:\Program Files\Autodesk Topobase Client 2009\Template\Modules\Sample* (create the Sample folder if needed). Click Save.

17 In the Template Description dialog box click OK to accept the English measurement default.

18 Click the Topobase Administrator icon.

19 Click Drawing Template in the left pane. A form labeled Drawing Template is displayed on the right.

20 Click Browse.

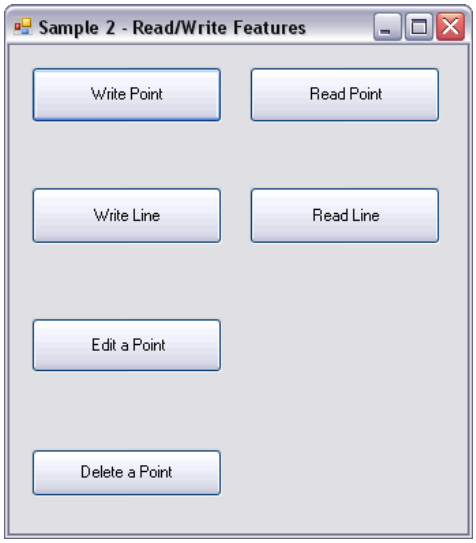
21 In the Open dialog box browse to *C:\Program Files\Autodesk Topobase Client 2009\Template\Modules\Sample*. Select *TBSAMPLE.dwt*. Click Open.

22 Bring Topobase to the foreground. Click Generate Graphic.

Sample 02 - Read/Write Features

Purpose

This sample demonstrates reading, writing, editing, and deleting point features in a Topobase workspace within an application. This sample uses the TBSAMPLE workspace created by the Sample 01 application.



Procedure

The write, edit and delete operations are followed by read operations for verification. This procedure also tells you how to monitor the effects of the application operations in the Oracle database using Oracle SQL*Plus commands.

To use this sample:

1 Build the sample. The plugin (*VBSample02.dll* or *CSSample02.dll*) will be placed in the Topobase Client *bin* directory. The *.tbp* file will also be copied to the Topobase Client *bin* directory.

2 Start Topobase, opening the TBSAMPLE workspace.

3 Click the application toolbar button with the tooltip Sample 02 - Read Write Features. The sample's dialog box will be displayed.

4 Click the Write Point button.

NOTE A form with a progress bar is displayed. This indicates that a connection is being made to the Topobase workspace. At the end of the write operation the connection is closed. Each time you click a button on this form a connection to a workspace is opened and closed. The descriptions of the other operations do not repeat this information.

---

5 In the XY Input dialog box type a number in the text box labeled X and in the text box labeled Y and click OK. The result is that an untitled message box is displayed telling the feature ID number of the point that you just added. Click OK.

To display the results of this operation in the TBSAMPLE user tables in your Oracle instance, enter the following commands at a DOS command prompt:

```
sqlplus /nolog
connect tbsample/avs
select fid,geom from mypoint;
```

6 Click the Read Point button. Type the feature ID number returned by the write point operation into the text box labeled FID in the Feature Class ID dialog box. Click OK. A message box is displayed showing the X and Y values that you specified during the write point operation. Click OK.

7 Click the Edit a Point button. Type the feature ID number returned by the write point operation into the text box labeled FID in the Feature Class ID dialog box. Click OK. In the XY Input dialog box type a number in the text box labeled X and in the text box labeled Y and click OK. Use Read Point from the previous step again to verify that you have changed the values of this feature.

8 Click the Delete a Point button. Click the OK button in the message box that says TBConnectionState: Successful. Type the feature ID number returned by the write point operation into the text box labeled FID in the Feature Class ID dialog box. Click OK. Click the OK button in the

message box that says Deletion of MYPOINT feature with FID = <number> succeeded.

9 Click the Read Point button. Type the feature ID number returned by the write point operation into the text box labeled FID in the Feature Class ID dialog box. Click OK. Click the OK button in the message box that says Feature with this FID not found.

At any time during the execution of this sample, right-click MyPoint and select Show Form to display the points and lines created by this sample.

Sample 03 - Oracle Data Provider (ODP) .Net

Purpose

This sample shows how to access data using the Oracle Data Provider (ODP) and Microsoft's .Net technology. The code features the use of the following classes:

- `Topobase.Data.TBConnection`
- `Topobase.Data.Provider.Command`
- `Topobase.Data.Provider.DataReader`
- `Topobase.Data.Provider.DataAdapter`
- `Topobase.Data.DBTable`
- `Topobase.Data.Row`
- `Topobase.Data.Attribute`

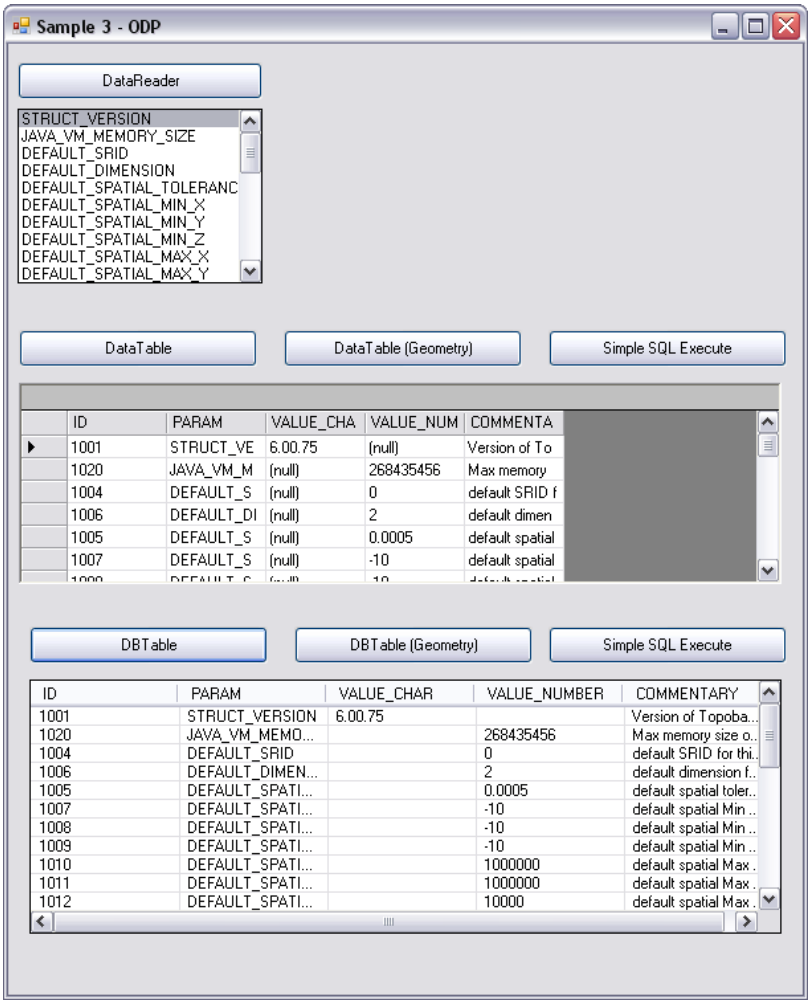
Procedure

To use this sample:

1 Build the sample. The plugin (*VBSample03.dll* or *CSSample03.dll*) will be placed in the Topobase Client *bin* directory. The *.tbp* file will also be copied to the Topobase Client *bin* directory.

2 Start Topobase, opening the TBSAMPLE workspace.

3 Click the Setup ➤ Sample 03 - ODP menu item. The sample's dialog box will be displayed.



This dialog box and associated buttons illustrate how you can programatically retrieve and modify data. For example, click DataReader to retrieve data from the database. Other buttons perform similar tasks. For more information about the functionality and sequence of calls, review the sample code.

Sample 10 - Document Context Menu

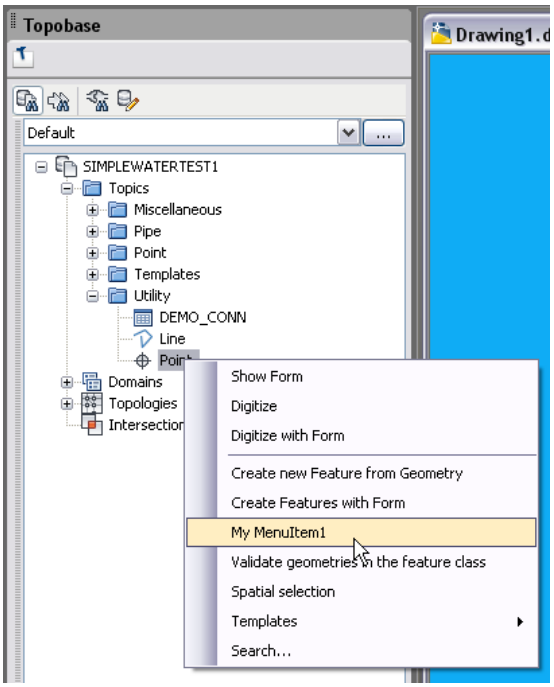
Purpose

This sample shows how to add a menu item to the context menu of the Feature Explorer tree. This sample uses the TBSAMPLE workspace created by the Sample 01 application.

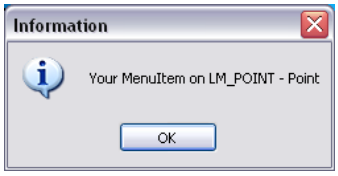
Procedure

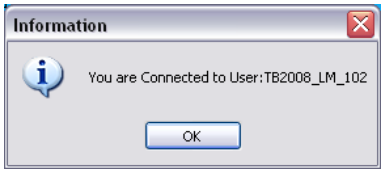
To use this sample:

1. Build the sample. The plugin (*VBSample10.dll* or *CSSample10.dll*) will be placed in the Topobase Client *bin* directory. The *.tbp* file will also be copied to the Topobase Client *bin* directory.
2. Start Topobase and open the `TBSAMPLE` workspace.
3. Right-click MyPoint. A new My MenuItem1 menu item is displayed.



4 If you click the menu item, the event handler is triggered.





Sample 12 - Document Toolbar Button

Purpose

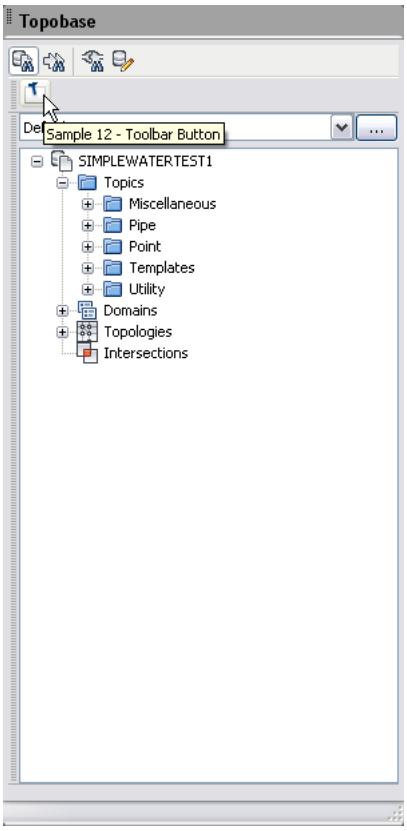
This sample shows how to add a toolbar button to a document.

Procedure

To use this sample:

1 Build the sample. The plugin (*VBSample12.dll* or *CSSample12.dll*) will be placed in the Topobase Client *bin* directory. The *.tbp* file will also be copied to the Topobase Client *bin* directory.

2 Start Topobase and open any workspace. A new toolbar button with the caption Sample 12 - Toolbar Button is displayed in a document-level toolbar in the Topobase task pane.



3 If you click the button, the event handler is triggered.

Sample 13 - Dialog Toolbar Button

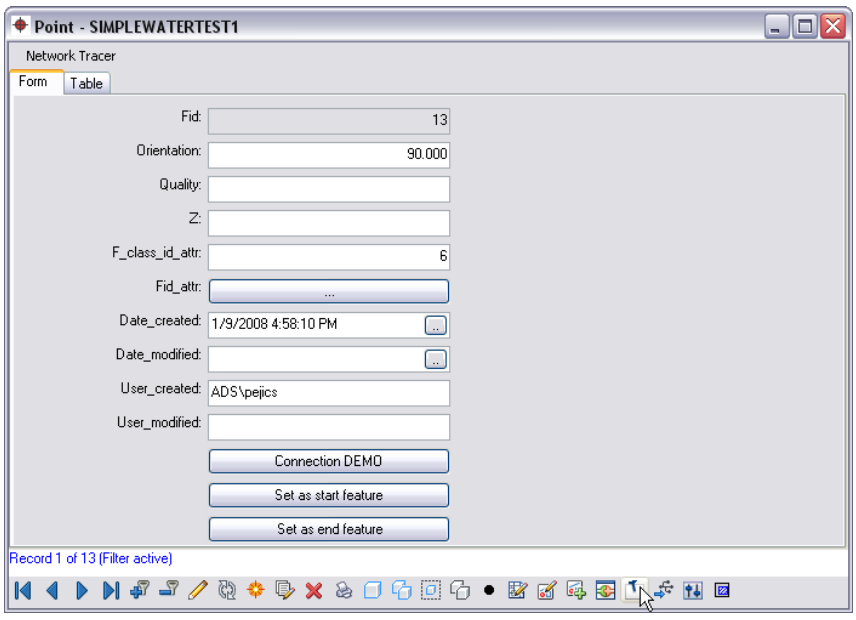
Purpose

This sample shows how to add a toolbar button to a generic dialog box.

Procedure

To use this sample:

1. Build the sample. The plugin (*VBSample13.dll* or *CSSample13.dll*) will be placed in the Topobase Client *bin* directory. The *.tbp* file will also be copied to the Topobase Client *bin* directory.
2. Start Topobase and open any workspace.
3. Right-click any feature and select Show Form. A new button with the caption Sample 13 - Toolbar Button is displayed on the dialog box's bottom toolbar.



4. If you click the button, the event handler is triggered.

Sample 14 - Main Menu

Purpose

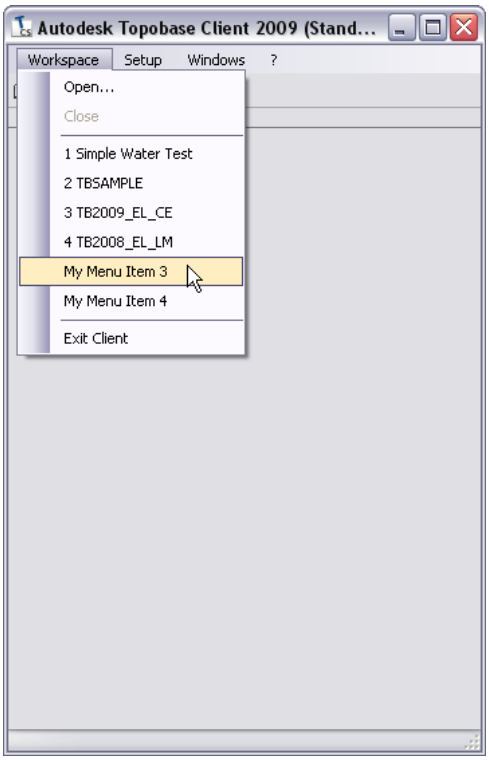
This sample shows how to add menu items to the main menu of the Topobase application.

Procedure

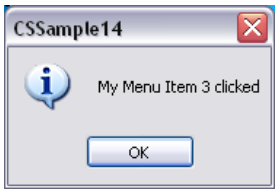
To use this sample:

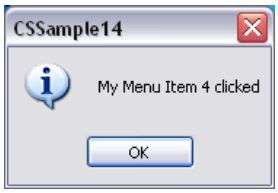
1. Build the sample. The plugin (*VBSample14.dll* or *CSSample14.dll*) will be placed in the Topobase Client *bin* directory. The *.tbp* file will also be copied to the Topobase Client *bin* directory.
2. Start either the Topobase Stand-Alone client or the desktop client without a workspace loaded. Two new menu items (My Menu Item 3 and My Menu Item 4) are added to the Workspace menu of the Topobase task pane.

NOTE If you run the Topobase desktop client with a workspace loaded, the application menu will not be shown, and the custom menus will not be seen by the user.

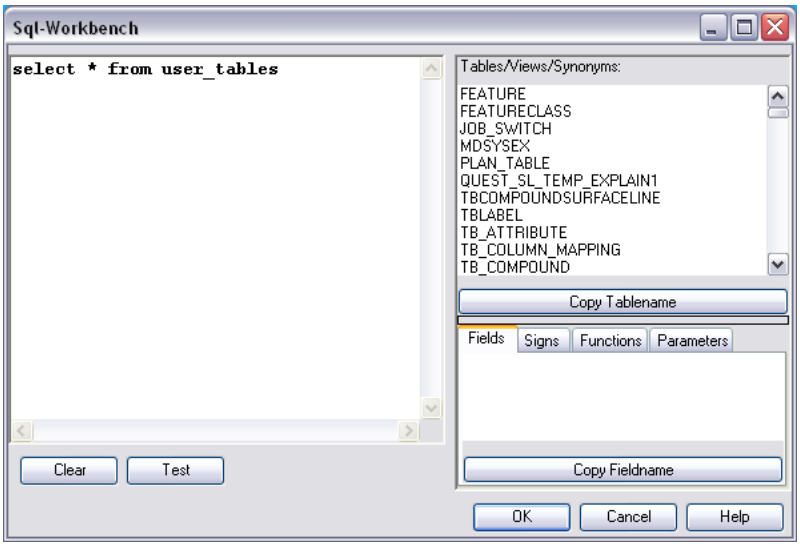


3 If you click the menu items, the corresponding event handler is triggered.





My Menu Item 4 opens the SQL Workbench.



Sample 15 - Main Toolbar

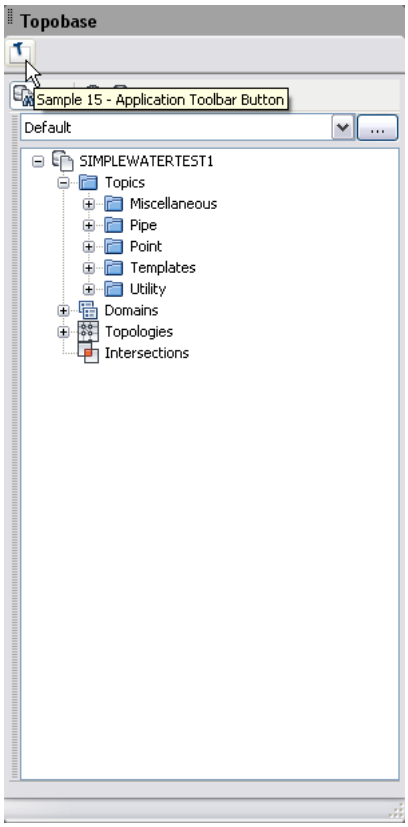
Purpose

This sample shows how to add a toolbar button to the main application toolbar.

Procedure

To use this sample:

1. Build the sample. The plugin (*VBSample15.dll* or *CSSample15.dll*) will be placed in the Topobase Client *bin* directory. The *.tbp* file will also be copied to the Topobase Client *bin* directory.
2. Start Topobase. A new button with the caption Sample 15 - Application Toolbar Button is displayed in a application-level toolbar in the Topobase task pane.



3 If you click the button, the event handler is triggered.

Sample 16 - Dialog Button

Purpose

This sample shows how to add a toolbar button to a generic dialog box. This sample uses the TBSAMPLE workspace created by the Sample 01 application.

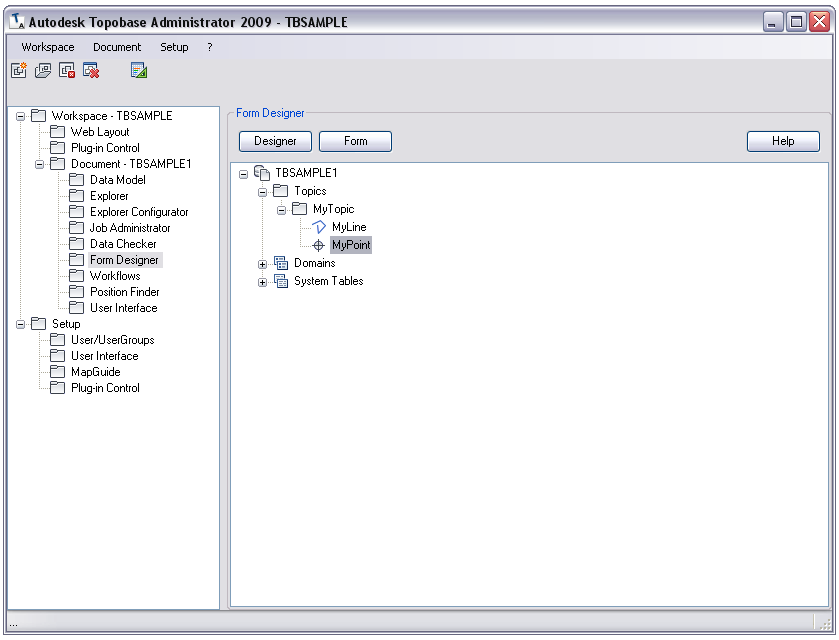
Procedure

To use this sample:

1. If you have not already done so, run Sample 01 to create the `TBSAMPLE` workspace. (See Sample 01 - Create Structure (page 149).)
2. Add a control to the generic dialog box for MyPoint.

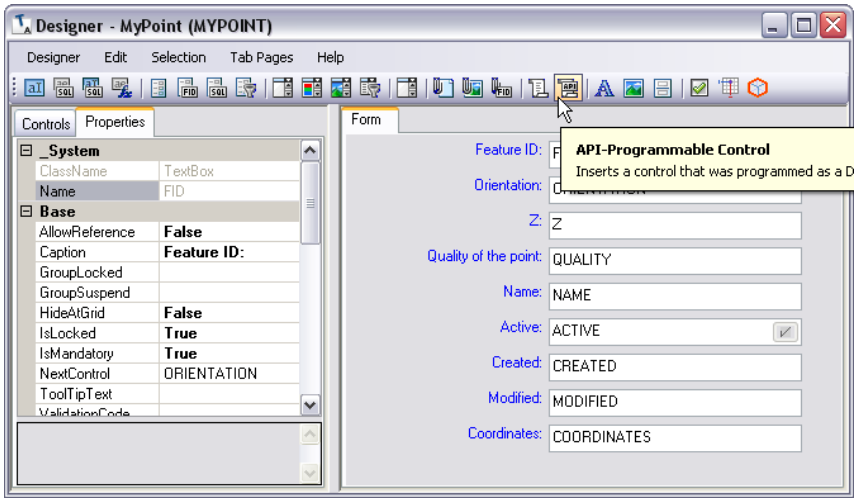
NOTE You must do this *first* before building the sample. If you do not, an error message "Control $myButton1 not found in Dialog" will be displayed.

1. Start the Topobase Administrator and open the `TBSAMPLE` workspace.

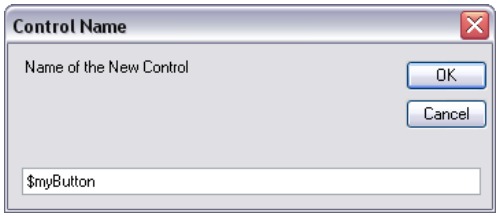


2 In the Form Designer, select MyPoint, then click Designer.

3 Click the API-Symbol icon.

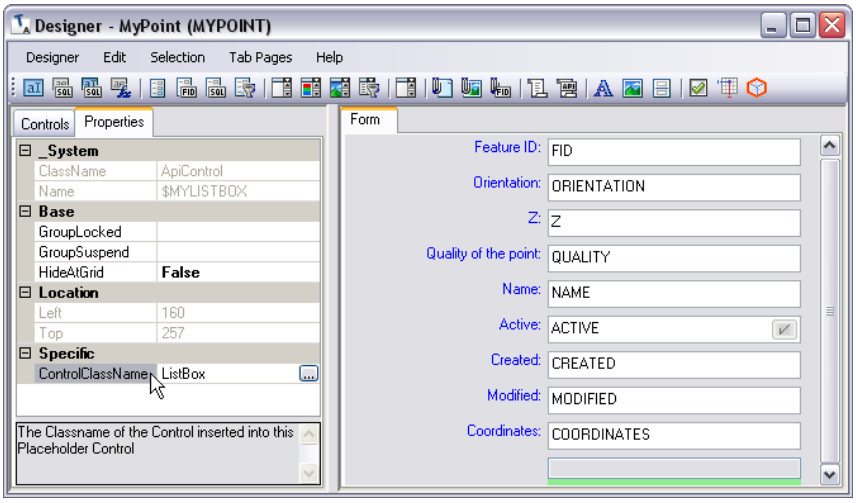


4 Give the new control the name $myButton1.



5 Click the API-Symbol again to create another control and give the new control the name $myListBox1.

6 Select that new control on the right, select ControlClassName on the left, and click the browse button that is displayed beside ControlClassName.



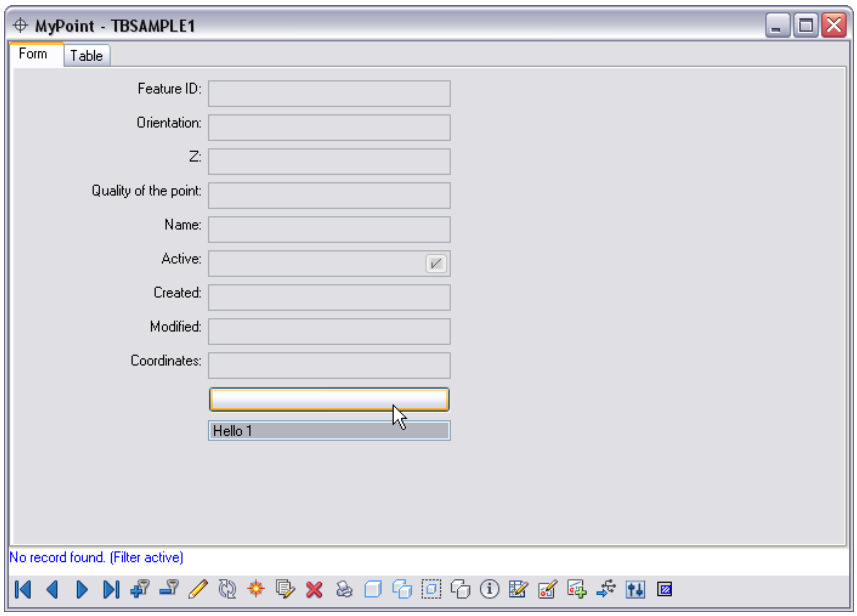
Set the Classname to ListBox.

7 Close the Designer and exit the Administrator.

3 Build the sample. The plugin (*VBSample16.dll* or *CSSample16.dll*) will be placed in the Topobase Client *bin* directory. The *.tbp* file will also be copied to the Topobase Client *bin* directory.

4 Start Topobase and open the TBSAMPLE workspace.

5 Right-click MyPoint and select Show Form. Two new controls are displayed on the dialog box.



Sample 17 - Dialog Menu

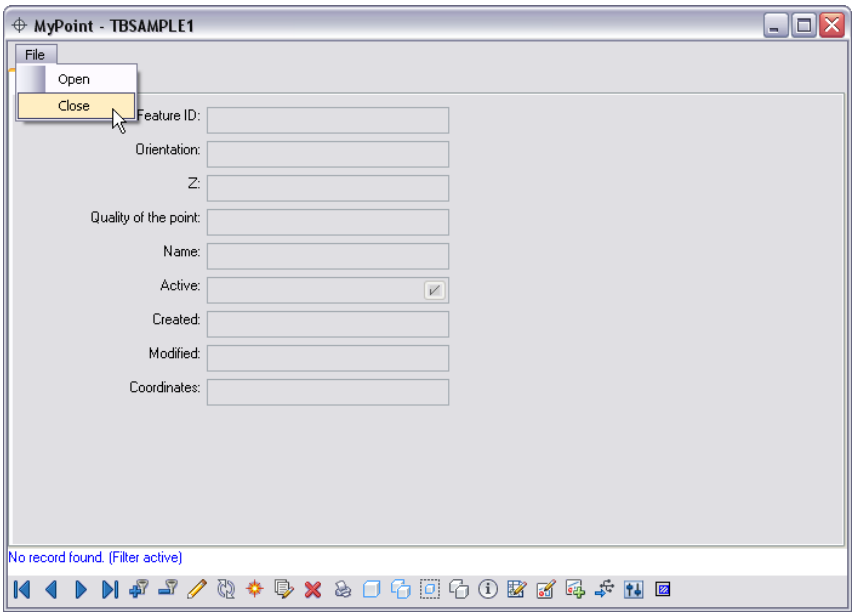
Purpose

This sample shows how to add a menu to a generic dialog box. This sample uses the TBSAMPLE workspace created by the Sample 01 application.

Procedure

To use this sample:

1. Build the sample. The plugin (*VBSample17.dll* or *CSSample17.dll*) will be placed in the Topobase Client *bin* directory. The *.tbp* file will also be copied to the Topobase Client *bin* directory.
2. Start Topobase and open the `TBSAMPLE` workspace.
3. Right-click MyPoint and select Show Form. A new File menu is added to the dialog box.



Sample 18 - Dialog Events

Purpose

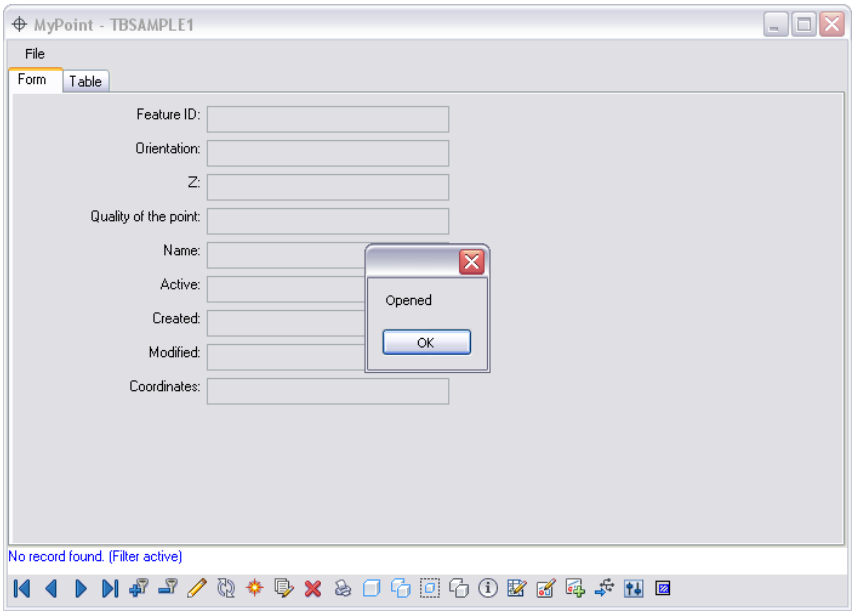
This sample shows how to detect events that are triggered as the user interacts with dialog boxes. This sample uses the TBSAMPLE workspace created by the Sample 01 application.

Procedure

To use this sample:

1. Build the sample. The plugin (*VBSample18.dll* or *CSSample18.dll*) will be placed in the Topobase Client *bin* directory. The *.tbp* file will also be copied to the Topobase Client *bin* directory.
2. Start Topobase and open the `TBSAMPLE` workspace.

3 Right-click MyPoint and select Show Form. The sample will display messages when various events are triggered.



TIP After completing this sample, remove the Sample 18 *.dll*, *.tbp*, and *.xml* from the Topobase *bin* directory to avoid extra dialog boxes.

Sample 19 - Input Box, Confirm Box, and Multiple Input Box

Purpose

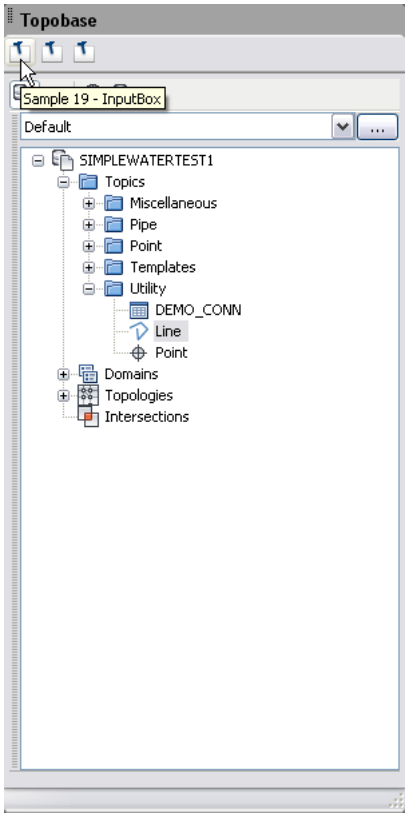
This sample shows how to use input boxes, confirmation boxes, and multiple input boxes. These work on both the Client and the Web interface.

NOTE The default Windows InputBox does not work for web applications.

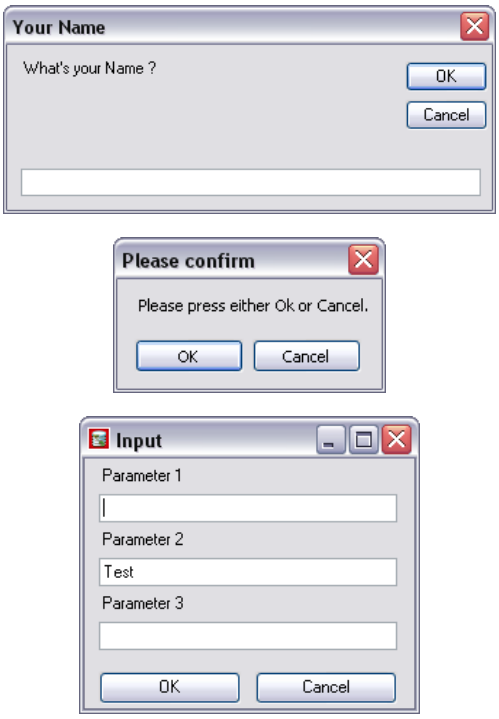
Procedure

To use this sample:

1. Build the sample. The plugin (*VBSample19.dll* or *CSSample19.dll*) will be placed in the Topobase Client *bin* directory. The *.tbp* file will also be copied to the Topobase Client *bin* directory.
2. Start Topobase.
3. Select one of the three buttons that the plugin added to the application toolbar of the Topobase task pane.



The sample displays the different kinds of dialog boxes with interaction for each.



Sample 20 - Forms and Controls

Purpose

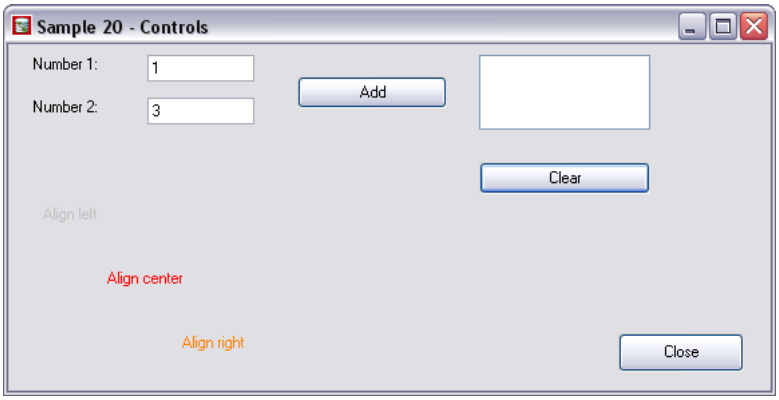
This sample shows how to create forms and controls.

Procedure

To use this sample:

1. Build the sample. The plugin (*VBSample20.dll* or *CSSample20.dll*) will be placed in the Topobase Client *bin* directory. The *.tbp* file will also be copied to the Topobase Client *bin* directory.
2. Start Topobase.

3 Click the application toolbar button with the caption Sample 20 - Forms and Controls. The sample's dialog box shows the various controls that you can use.



Sample 22 - Document Flyin

Purpose

FlyIns are components of a Topobase task pane that can either be docked in (that is, included in the application window) or docked out (that is, independently "flying" on the desktop). For example, the following elements of the Topobase user interface are flyins:

- Document Explorer
- Job Management
- Workflow Explorer

For each document and user group you can define which flyins are available.

This sample shows how to extend a document with two flyins.

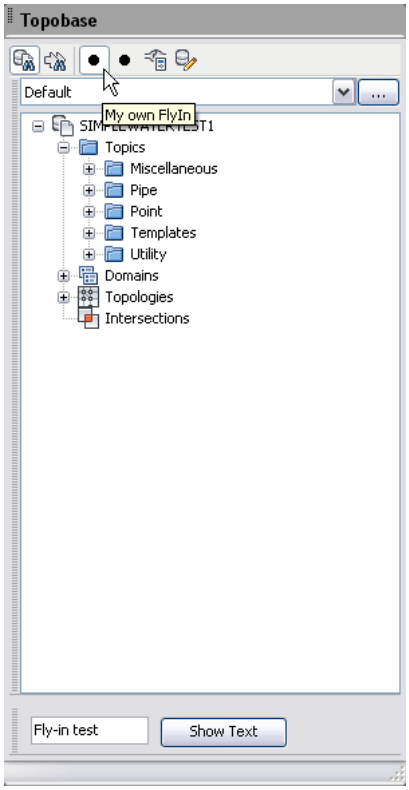
Procedure

To use this sample:

1 Build the sample. The flyin (*VBSample22.dll* or *CSSample22.dll*) will be placed in the Topobase Client *bin* directory. The *.tbp* file will also be copied to the Topobase Client *bin* directory.

2 Start Topobase and open any workspace. Two new buttons are added to the document toolbar in the Topobase task pane. These buttons launch the two flyins: My Own Flyin and My Personal Desktop Flyin.

3 Select one of the toolbar buttons. A docked flyin is shown at the bottom of the Topobase task pane. Only one of the added flyins can be shown at one time.



NOTE For use in the Web mode, the flyin's dock state should not be set. For use in the Client mode, only one control can have the dock state set to "Full". Normally, this is the Feature Explorer. However, if you remove the Feature Explorer and replace it with your own navigator then you should set its dock state to "Full".

Sample 25 - Progress Bar

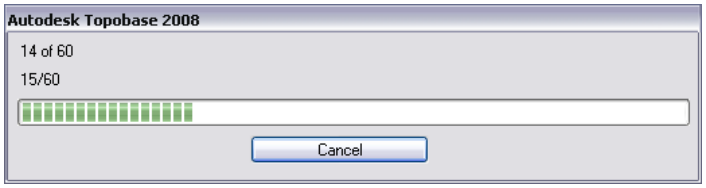
Purpose

This sample shows how to display a progress bar while your application performs a lengthy task.

Procedure

To use this sample:

1. Build the sample. The plugin (*VBSample22.dll* or *CSSample22.dll*) will be placed in the Topobase Client *bin* directory. The *.tbp* file will also be copied to the Topobase Client *bin* directory.
2. Start Topobase and open any workspace.
3. Click the application toolbar button with the caption Sample 25 - Progress Bar display a progress bar.



Sample 28 - List Box With Context Menu

Purpose

This sample shows how to display a list box with a context menu for each item in the list.

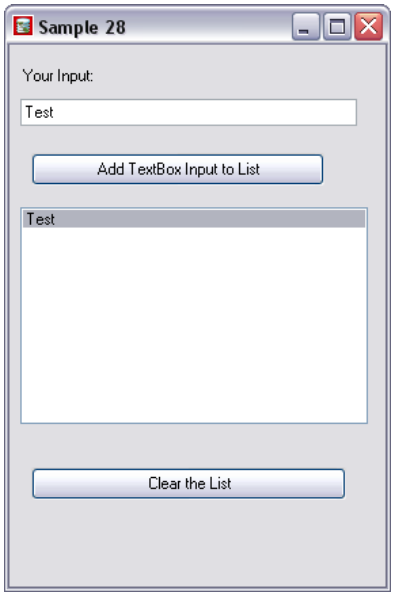
Procedure

To use this sample:

1 Build the sample. The plugin (*VBSample28.dll* or *CSSample28.dll*) will be placed in the Topobase Client *bin* directory. The *.tbp* file will also be copied to the Topobase Client *bin* directory.

2 Start Topobase.

3 Click the application toolbar button with the caption Sample 28 - Listbox with ContextMenu. The sample's dialog box is displayed.



Sample 29 - Tree View

Purpose

This sample shows how to use a tree view control.

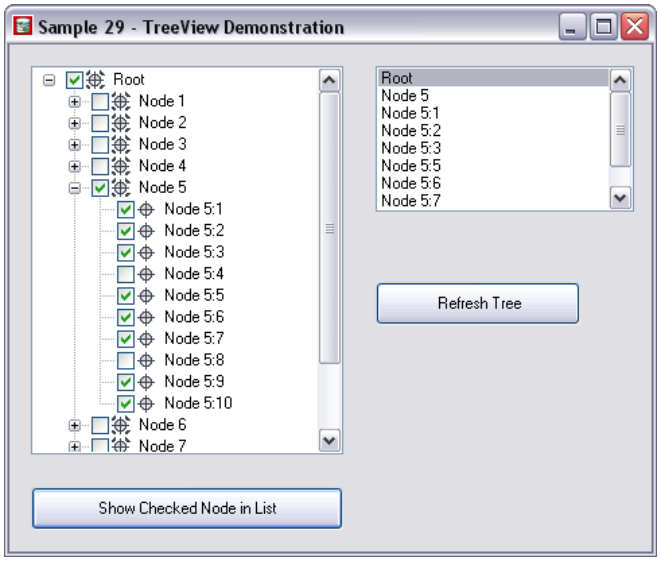
Procedure

To use this sample:

1 Build the sample. The plugin (*VBSample29.dll* or *CSSample29.dll*) will be placed in the Topobase Client *bin* directory. The *.tbp* file will also be copied to the Topobase Client *bin* directory.

2 Start Topobase.

3 Click the application toolbar button with the caption Sample 29 - TreeView Control. The sample's dialog box is displayed.



Sample 30 - Map Button

Purpose

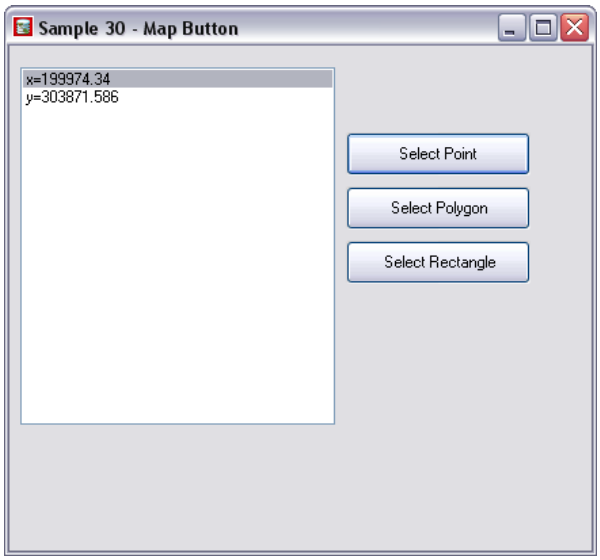
This sample shows how to retrieve the coordinates defining features for three feature types: point, polygon, and rectangle.

Procedure

To use this sample:

1. Build the sample. The plugin (*VBSample30.dll* or *CSSample30.dll*) will be placed in the Topobase Client *bin* directory. The *.tbp* file will also be copied to the Topobase Client *bin* directory.
2. Start Topobase and open the land management demonstration workspace (for example, TB2009_LM_102) whose creation is described in the *Autodesk Topobase Installation and Configuration Guide*.
3. Click Generate Graphic.
4. Click the document toolbar button with the caption Sample 30 - MapButton Control.

   The sample's dialog box is displayed.



5. Click Select Point. Move the cursor over the generated graphic and click. Bring the Sample 30 dialog to the foreground. The x and y values of the point's coordinates are displayed in the list box.
6. Click Select Polygon. The Topobase window is moved to the foreground. Move the cursor over the generated graphic and click. Move the cursor and click. Move the cursor and click. Now right-click to display a popup menu and select Close. You have defined a triangle. Bring the Sample 30 dialog to the foreground. There are four XY coordinates in the list box. The first and last coordinates have the same value.

7 Click Select Rectangle. The Topobase window is moved to the foreground. Move the cursor over the generated graphic and click. Move the cursor and click. Bring the Sample 30 dialog to the foreground. There are MaxX, MaxY, MinX, and MinY values in the list box.

Sample 31 - List Box

Purpose

This sample shows how to use the list box control.

Procedure

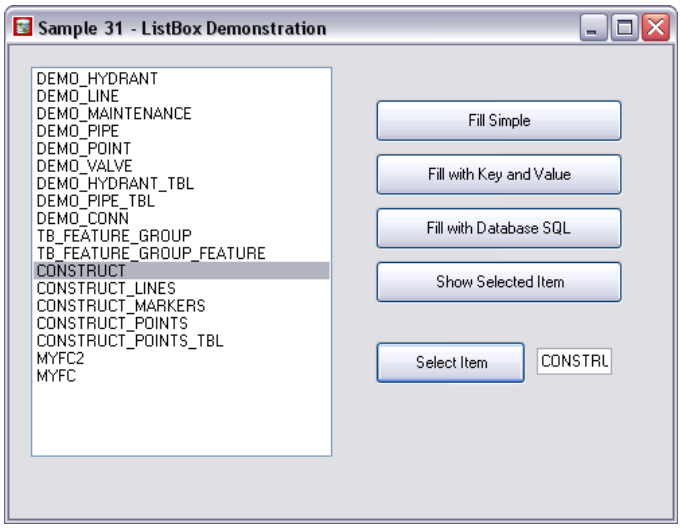
To use this sample:

1 Build the sample. The plugin (*VBSample31.dll* or *CSSample31.dll*) will be placed in the Topobase Client *bin* directory. The *.tbp* file will also be copied to the Topobase Client *bin* directory.

2 Start the Topobase desktop or web client and open any workspace.

3 Click the desktop toolbar button with the caption Sample 31 - Listbox.

4 The sample's dialog box is displayed allowing you to experiment with the functionality of the buttons.



Sample 32 - Highlight

Purpose

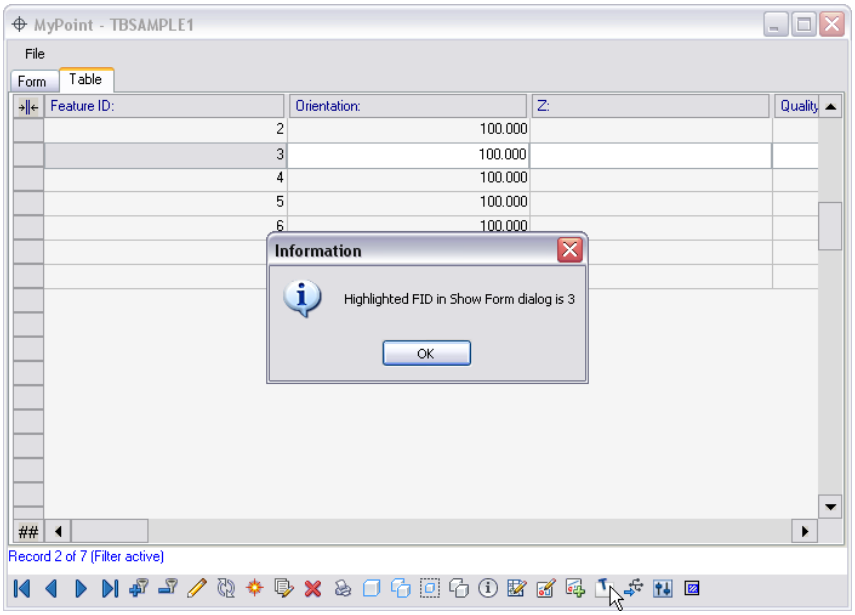
This sample shows how to get information about currently selected items in a form. This sample uses the TBSAMPLE workspace created by the Sample 01 application.

Procedure

To use this sample:

1. Build the sample. The plugin (*VBSample32.dll* or *CSSample32.dll*) will be placed in the Topobase Client *bin* directory. The *.tbp* file will also be copied to the Topobase Client *bin* directory.
2. Start Topobase and open the `TBSAMPLE` workspace.
3. Right-click MyPoint and select Show Form. The points form is displayed.
4. Select the Table tab.
5. Select one of the rows in the table.

6 Click the icon labeled “Sample 32 - Highlighting” in the bottom toolbar.



The sample will display a message box with information about the FID of the selected row. It will then highlight the selected point in the graphics pane.

Sample 33 - Information

Purpose

This sample shows how to use the Feature Information icon to display the Show Form dialog box with a selected feature displayed in the dialog box. This sample uses the TBSAMPLE workspace created by the Sample 01 application.

Procedure

To use this sample:

1. Build the sample. The plugin (*VBSample33.dll* or *CSSample33.dll*) will be placed in the Topobase Client *bin* directory. The *.tbp* file will also be copied to the Topobase Client *bin* directory.
2. Start Topobase and open the `TBSAMPLE` workspace.

   NOTE If you have not already done so, use Sample 02 to add a couple of points and a line to the TBSAMPLE workspace. Or, add them manually through by right-clicking MyPoint and MyLine, choosing Show Form, and adding records.

3. Click Generate Graphic.

   TIP If needed, you can right-click on a feature in the Display Manager pane and select Zoom to Extents to more easily view the object.

4. In the graphics pane click a point.
5. Click the Attributes item in the ribbon or in the context menu of the point. The message box is displayed with the text Sample 33 OnInfo handler running: e.DialogName: MYPOINT. Click OK. A show form dialog box labeled MyPoint - TBSAMPLE is displayed with the tab labeled Form selected. This tab contains information about the selected point.
6. In the graphics pane click a line.
7. Click the feature information button. A message box is displayed with the text Sample 33 OnInfo handler running: e.DialogName: MYLINE. Click OK. A show form dialog box labeled MyLine - TBSAMPLE is displayed with the tab labeled Table selected. This tab contains information about the selected line.

Sample 34 - File Upload and Download

Purpose

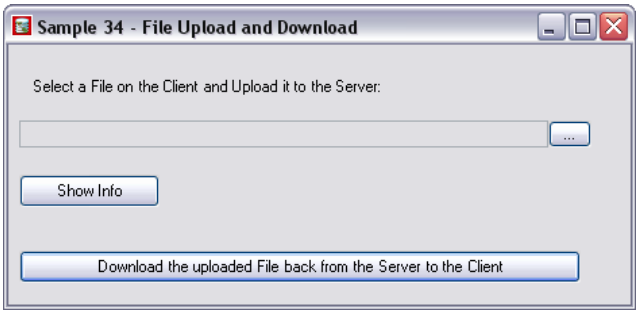
This sample shows how to upload files from the client to the server and download from the server to the client.

Procedure

To use this sample:

1. Build the sample. The plugin (*VBSample34.dll* or *CSSample34.dll*) will be placed in the Topobase Client *bin* directory. The *.tbp* file will also be copied to the Topobase Client *bin* directory.
2. Start the Topobase desktop or web client and open any workspace.
3. Click the document toolbar button with the caption Sample 34 - File Upload/Download .

   The upload/download dialog box is displayed. Select a file to "upload" and click Show Info. Note that in the client mode, the uploading and downloading of files to/from the server/client is unnecessary. This only applies when the client and server are different, such as in web mode.



Sample 37 - Windows Menu Items

Purpose

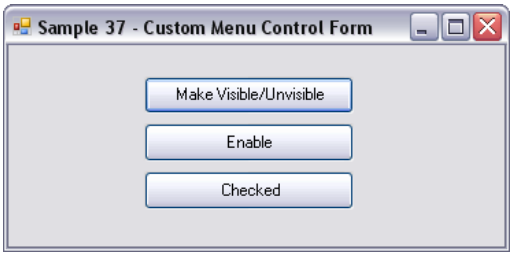
This sample shows how to manipulate menu items in the client.

NOTE All the properties for the `Topobase.Forms.MenuItem` class are safe under both Client and Web mode. This sample shows how to use extra properties (enabled, visible, checked) but the form can only be used in Client mode, not Web mode.

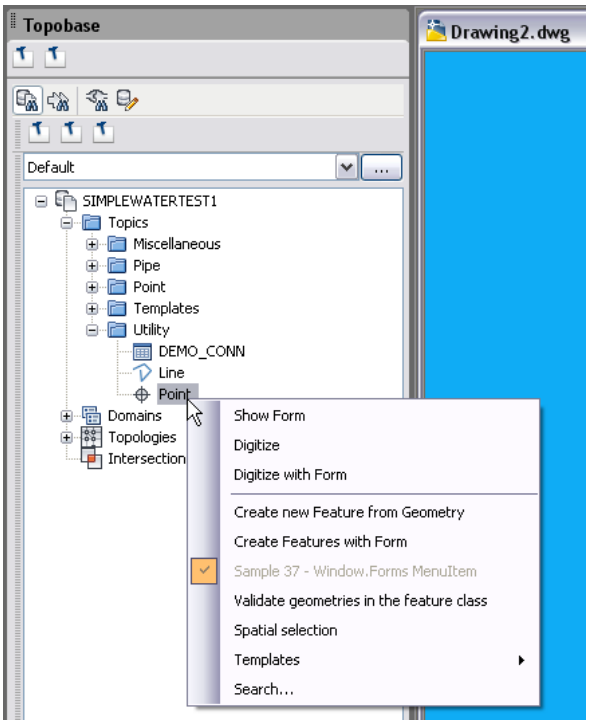
Procedure

To use this sample:

1. Build the sample. The plugin (*VBSample37.dll* or *CSSample37.dll*) will be placed in the Topobase Client *bin* directory. The *.tbp* file will also be copied to the Topobase Client *bin* directory.
2. Start Topobase and open a workspace which contains a Point feature type such as TBSAMPLE. The sample's dialog box opens automatically.



3. In the document explorer, expand the document tree until you can see a point feature class. Right-click on the feature class to display the context menu. Notice the Sample 37 - Windows.Forms MenuItem entry. Select this entry. An information dialog box is displayed with the text Menu Clicked.



4 Bring the sample's dialog box to the foreground. The buttons in this dialog box operate on the Sample 37 - Windows.Forms MenuItem entry. Click Enable. Display the MyPoint context menu again. The Sample 37 - Windows.Forms MenuItem entry is greyed out and inactive. Click Enable again. The Windows.Forms MenuItem entry is active again.

5 Bring the sample's dialog box to the foreground. Click Checked. Display the point feature class context menu again. The Sample 37 - Windows.Forms MenuItem entry now has a check mark prepended to it. Click Checked again. The check mark has been removed from the Sample 37 - Windows.Forms MenuItem entry.

Sample 40 - Create All Dialogs Boxes

Purpose

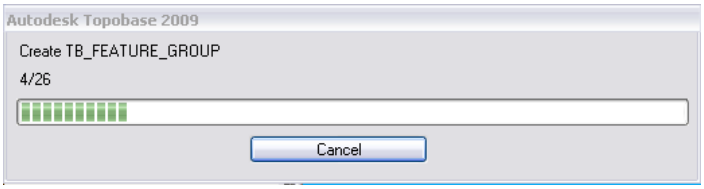
This sample shows how to create all the dialog boxes' entries in the database. This sample uses the TBSAMPLE workspace created by the Sample 01 application.

Normally, if you create a new Feature Class or Domain Table through the Topobase Administrator, the entries in various tables (such as `TB_GN_DIALOG`, `TB_GN_CONTROL` etc.) for the generic dialog boxes are not created until the user opens the dialog box for the first time. However, it can be useful to make sure that all entries are created in the database. Doing so enables you to batch-modify them all by using SQL without having to use the Form Designer manually.

Procedure

To use this sample:

1. Build the sample. The plugin (*VBSample40.dll* or *CSSample40.dll*) will be placed in the Topobase Client *bin* directory. The *.tbp* file will also be copied to the Topobase Client *bin* directory.
2. Start Topobase and open the `TBSAMPLE` workspace.
3. Click the document toolbar button with the caption Sample 40 - Create All Dialogs. A progress dialog box is displayed followed by a message box at the end of the process.



The code contains commented lines that, if restored, will display all dialogs as well as create them. This is disabled by default because of the large number of possible dialog boxes.

Sample 41 - Grid Control

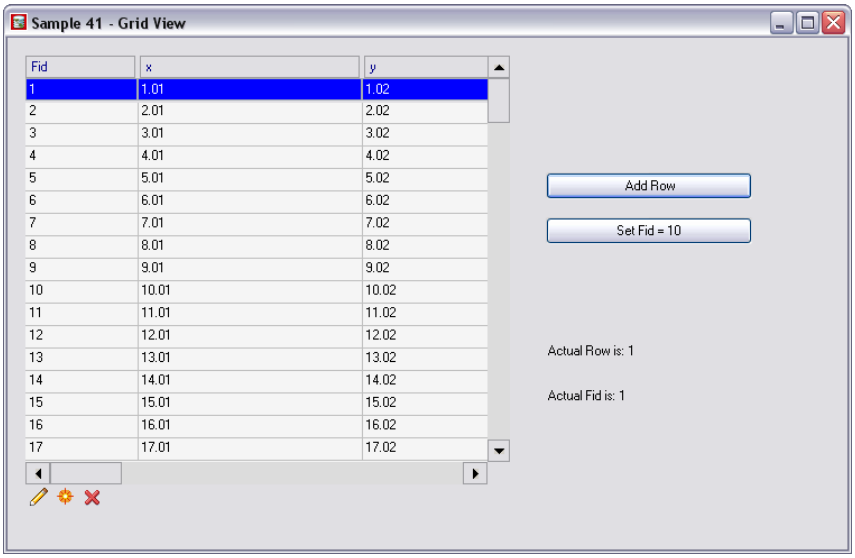
Purpose

This sample shows how to use a grid control.

Procedure

To use this sample:

1. Build the sample. The plugin (*VBSample41.dll* or *CSSample41.dll*) will be placed in the Topobase Client *bin* directory. The *.tbp* file will also be copied to the Topobase Client *bin* directory.
2. Start Topobase.
3. Click the application toolbar button with the caption Sample 41 - Grid View.



You can use the grid control, buttons, and icons in the dialog box to trigger various events and edit the data.

Sample 43 - Dialog Box Validation

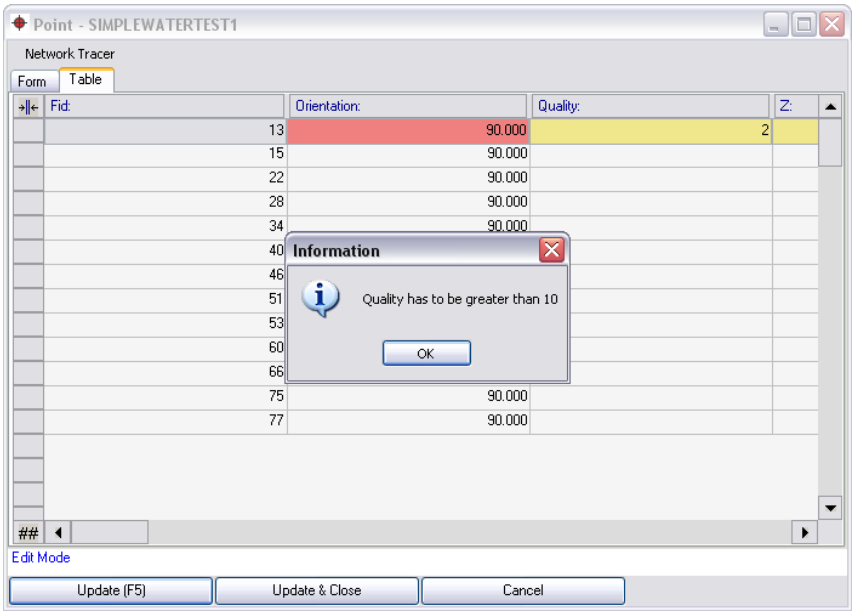
Purpose

This sample shows how to validate the data that a user enters in a generic dialog box. This sample uses the TBSAMPLE workspace created by the Sample 01 application.

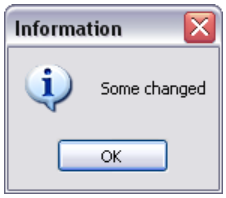
Procedure

To use this sample:

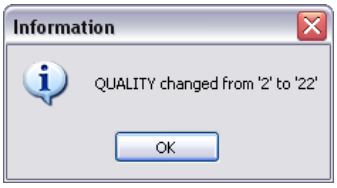
1. Build the sample. The plugin (*VBSample43.dll* or *CSSample43.dll*) will be placed in the Topobase Client *bin* directory. The *.tbp* file will also be copied to the Topobase Client *bin* directory.
2. Start Topobase and open the `TBSAMPLE` workspace.
3. In the Topobase pane, right-click MyPoint and select Show Form.
4. Change the Quality field to a value less than 10 and click Update. An Information dialog box is displayed with the text Quality Has To Be Greater Than 10. Click OK.



5 Change the Quality field to a value greater than 10 and click Update. An Information dialog box is displayed with the text Some Changed.



6 Click OK. An Information dialog box is displayed with the text Quality Changed From '<number>' To '<number>'. Click OK.



Sample 44 - Menus and Toolbars

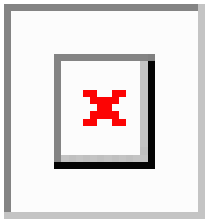
Purpose

This sample shows how to create menus and toolbars in a Topobase form.

Procedure

To use this sample:

1. Build the sample. The plugin (*VBSample44.dll* or *CSSample44.dll*) will be placed in the Topobase Client *bin* directory. The *.tbp* file will also be copied to the Topobase Client *bin* directory.
2. Start Topobase.
3. Click the application toolbar button with the caption Sample 44 - Form with Menu and Toolbar. A new form is displayed.



When you select the menu items or click the icons, message boxes will be displayed indicating which event took place.

Sample 45 - Matrix

Purpose

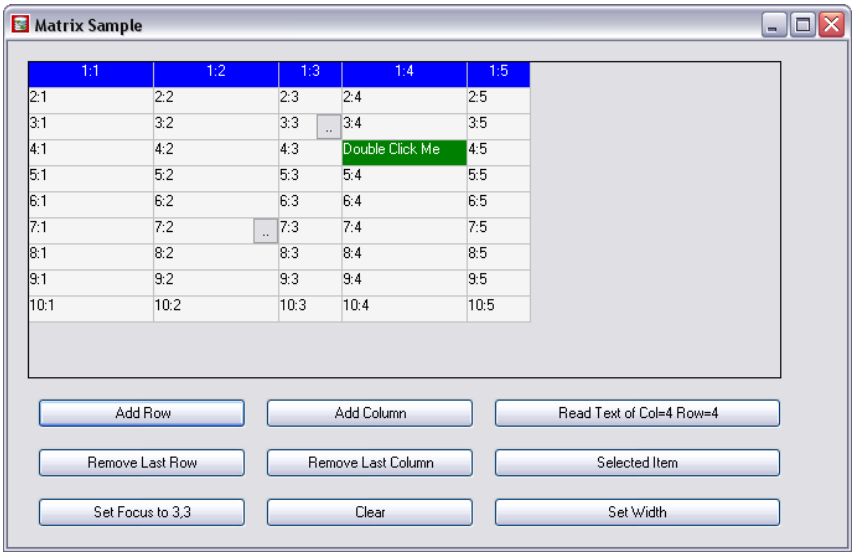
This sample shows how to use a matrix control.

Matrix controls are designed to work with small amounts of data - between 10 - 80 rows with up to 5 columns. These controls are cell oriented, so each cell can have a different color or be set to read-only. The Grid control, in contrast, is always in edit mode and is row oriented.

Procedure

To use this sample:

1. Build the sample. The plugin (*VBSample45.dll* or *CSSample45.dll*) will be placed in the Topobase Client *bin* directory. The *.tbp* file will also be copied to the Topobase Client *bin* directory.
2. Start Topobase.
3. Click the application toolbar button with the caption Sample 45 - Matrix. A new form is displayed.



The buttons show various ways you can programatically control the matrix.

Sample 46 - Interaction

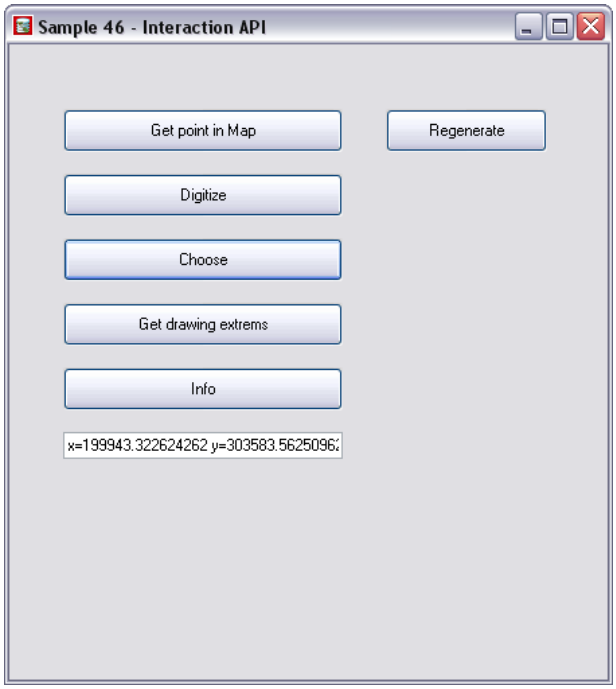
Purpose

This sample shows how a plugin can interact with a map. This sample uses the TBSAMPLE workspace created by the Sample 01 application.

Procedure

To use this sample:

1. Add some points and lines to the TBSAMPLE workspace. This can be done by using Sample 02 or within Topobase by right-clicking MyPoint and MyLine, choosing Show From, and adding records.
2. Build the sample. The plugin (*VBSample46.dll* or *CSSample46.dll*) will be placed in the Topobase Client *bin* directory. The *.tbp* file will also be copied to the Topobase Client *bin* directory.
3. Start Topobase and open the `TBSAMPLE` workspace.
4. Click Generate Graphic.
5. Specify an appropriate viewport and click Generate.
6. Click the document toolbar button with the caption Sample 46 - Web Interaction. A new form is displayed.



This form illustrates how you can interact with a map:

- Click GetPoint in Map and then click on a point in the generated graphic. Topobase displays an information dialog box with the text x=<number> y=<number>. Click OK. Topobase displays an information dialog box with the text Now you are back in the main thread. Click OK.
- Click Choose and then select a feature in the map. Topobase displays an information dialog box containing information about the feature you selected. Assuming you selected a point, the following text is displayed in the dialog box:

  ```
  Fid=<number>

  Class=MYPOINT

  FirstPointIndex=-1

  SecondPointIndex=-1
  ```

- Click Get Drawing Extremes. Topobase displays information about the extents of the map. It displays an information dialog box with the text Xmin=<number> Ymin=<number> Xmax=<number> Ymax=<number>. Click OK.
- Click Info and then click one point in the map. Press Enter. Topobase displays an information dialog box with the text Fid=<number>. Click OK.
- Click Info. Click anywhere in the map, drag the cursor to form a selection rectangle around a set of points and click. Press Enter. Topobase displays an information dialog box with the text Fid=<number>. Click OK. Topobase displays another information dialog box with the text Fid=<number>. Click OK. Topobase continues to display information dialog boxes until the feature class identifier for each of the points selected have all been displayed.

Sample 47 - Application Options

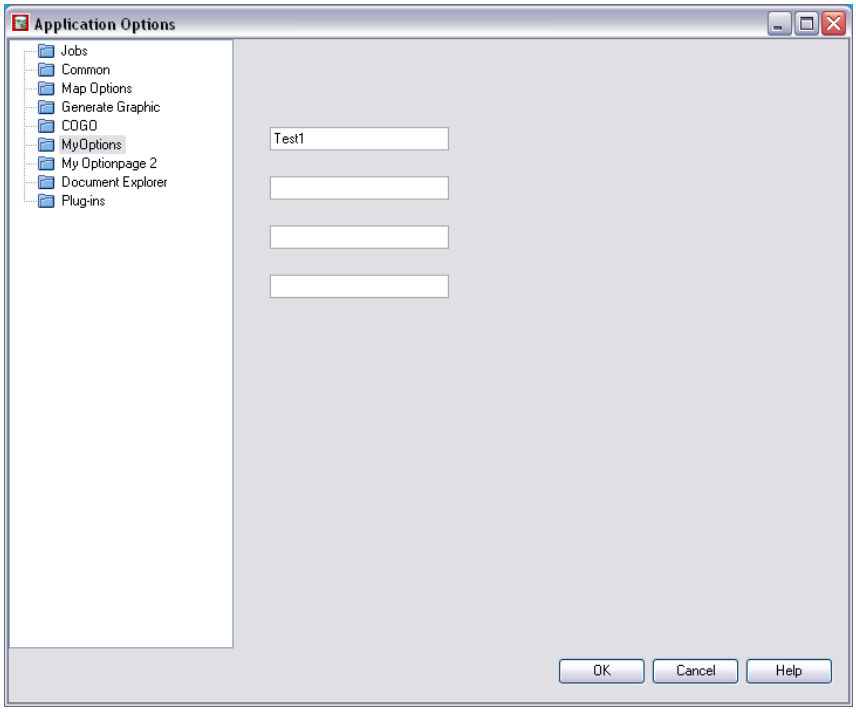
Purpose

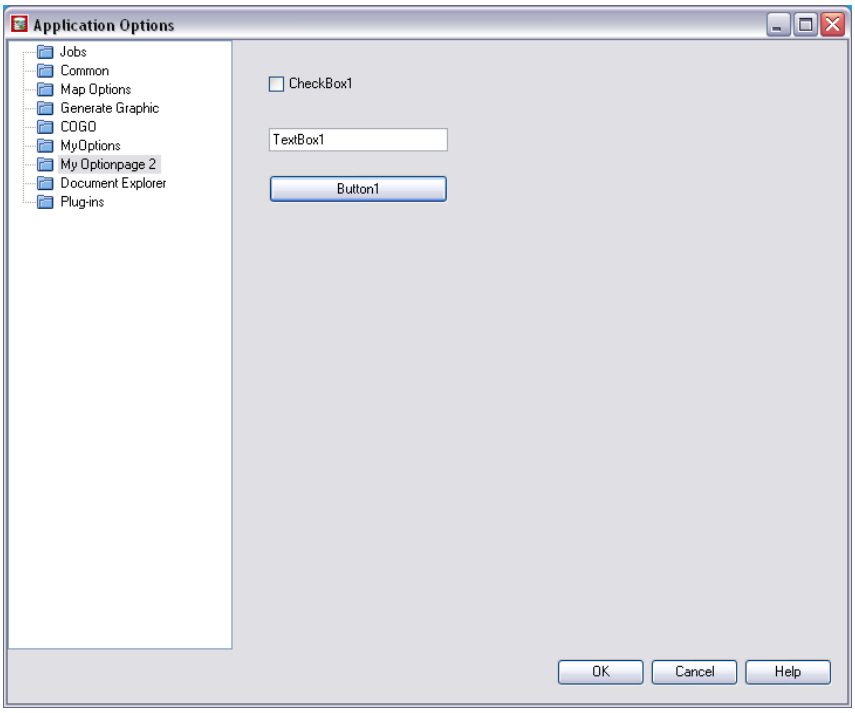
This sample shows how to create your own application-related option forms.

Procedure

To use this sample:

1. Build the sample. The plugin (*VBSample47.dll* or *CSSample47.dll*) will be placed in the Topobase Client *bin* directory. The *.tbp* file will also be copied to the Topobase Client *bin* directory.
2. Start Topobase.
3. Click Setup ➤ Options.
4. Click MyOptions or My Optionpage 2 in the tree. The new option forms are shown.





For a sample showing how to create *document* related options, see Sample 59 - Settings and Document Options (page 213).

Sample 48 - Treeview With Context Menu

Purpose

This sample shows how to create a treeview complete with context menus.

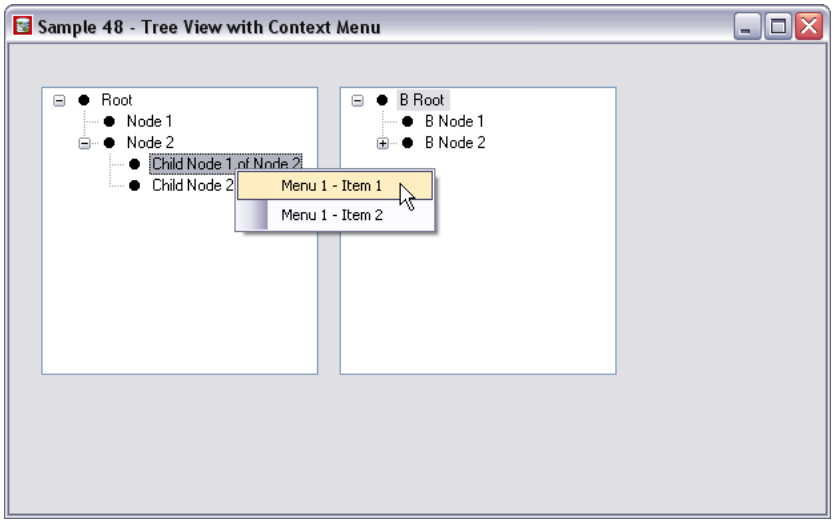
Procedure

To use this sample:

1 Build the sample. The plugin (*VBSample48.dll* or *CSSample48.dll*) will be placed in the Topobase Client *bin* directory. The *.tbp* file will also be copied to the Topobase Client *bin* directory.

2 Start Topobase.

3 Click the application toolbar button with the caption Sample 48 - TreeView with ContextMenu. The sample form with two treeview controls is displayed. Some of the nodes in the treeview controls have context menus.



Sample 50 - Date/Time Picker

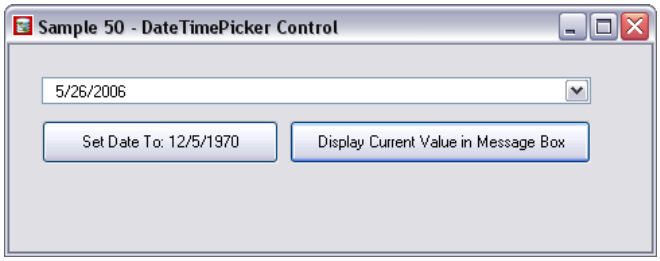
Purpose

This sample shows how to use the date/time control.

Procedure

To use this sample:

1. Build the sample. The plugin (*VBSample50.dll* or *CSSample50.dll*) will be placed in the Topobase Client *bin* directory. The *.tbp* file will also be copied to the Topobase Client *bin* directory.
2. Start Topobase.
3. Click the application toolbar button with the caption Sample 50 - DateTime Picker. The control is displayed.



Sample 51 - File and Directory Text Box

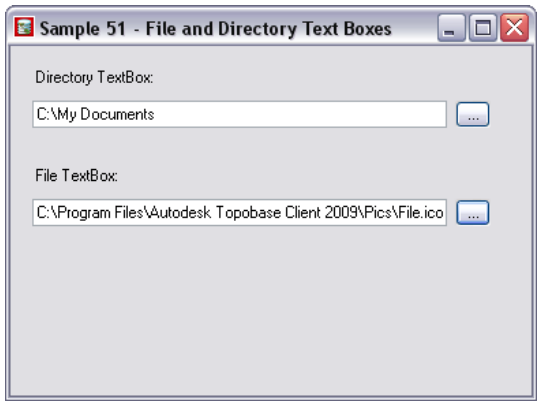
Purpose

This sample shows how to use the directory textbox control and the file textbox control.

Procedure

To use this sample:

1. Build the sample. The plugin (*VBSample51.dll* or *CSSample51.dll*) will be placed in the Topobase Client *bin* directory. The *.tbp* file will also be copied to the Topobase Client *bin* directory.
2. Start Topobase.
3. Click the application toolbar button with the caption Sample 51 - File And DirectoryTextBox. The new form is displayed, which enables you to select directories and files.



NOTE This control is not recommended for use in the web client. In the web client these controls show the files and directories of the web server machine, which may be a security risk. If you want to select a file on the web client machine, use the FileUploadBox control.

Sample 52 - Dock In Container

Purpose

This sample shows how to use the `Topobase.Forms.Application.Desktop.DialogDockInContainer` control and the `Topobase.Forms.Desktop.DockableForm` object.

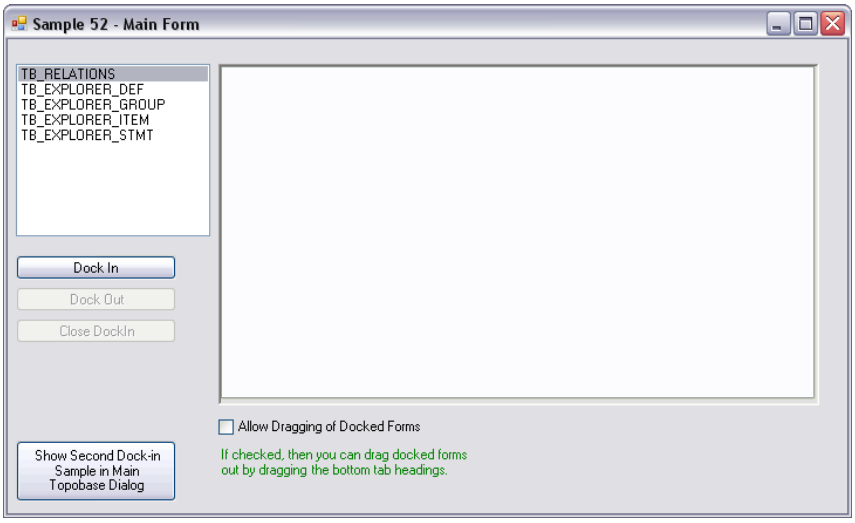
NOTE All docking controls only work in the Client mode. The `DockInContainer` control and the `DockableForm` class cannot be used in the Web mode.

Procedure

To use this sample:

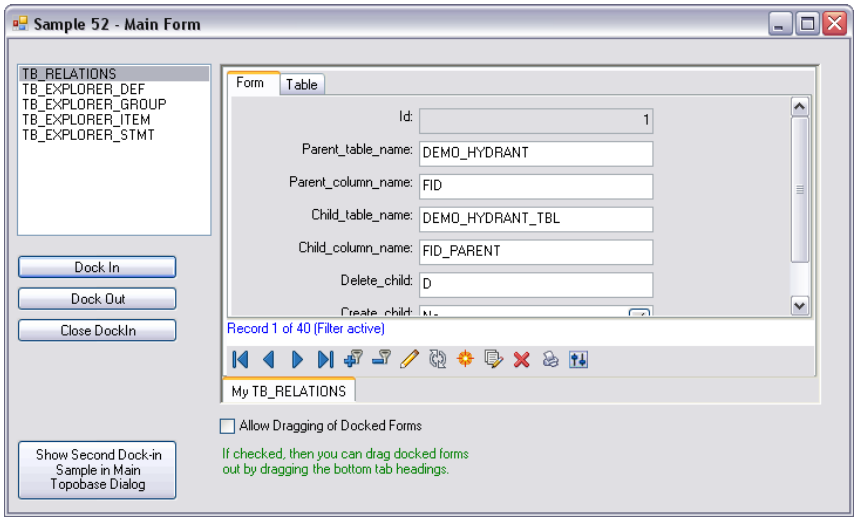
1. Build the sample. The plugin (*VBSample52.dll* or *CSSample52.dll*) will be placed in the Topobase Client *bin* directory. The *.tbp* file will also be copied to the Topobase Client *bin* directory.
2. Start Topobase and open any workspace.

3 Click the application toolbar button with the caption Sample 52 - DockIn Container. The new form is displayed.

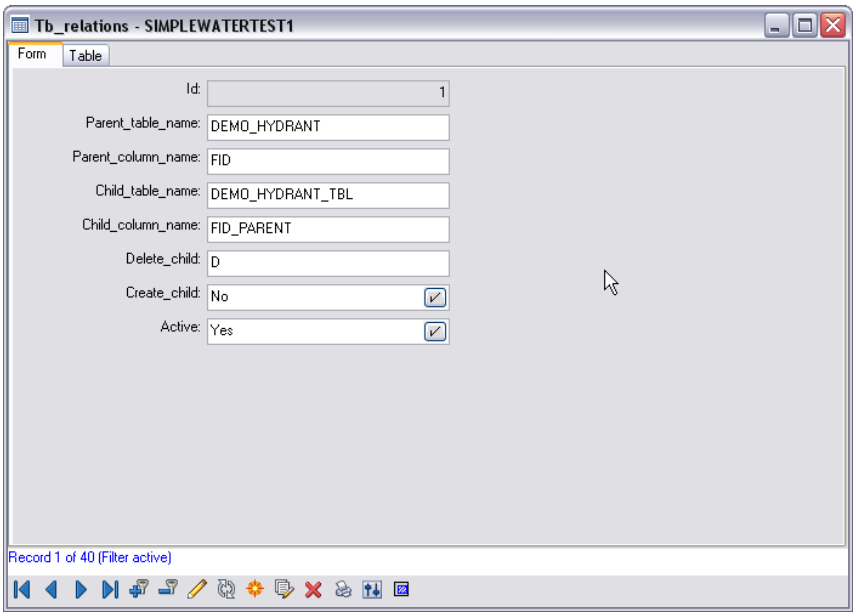


This illustrates different ways a form can be positioned:

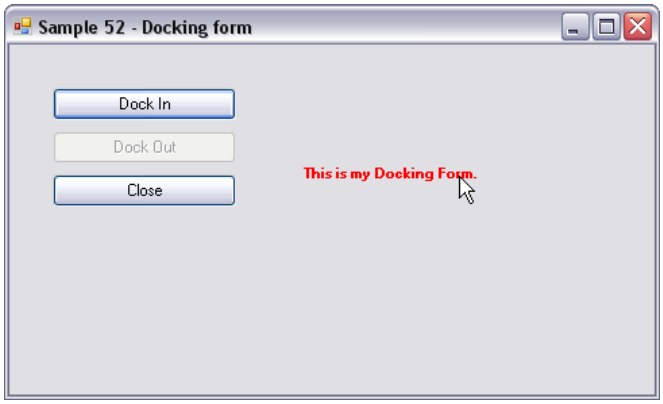
- Click Dock In to show the selected table docked into the form.



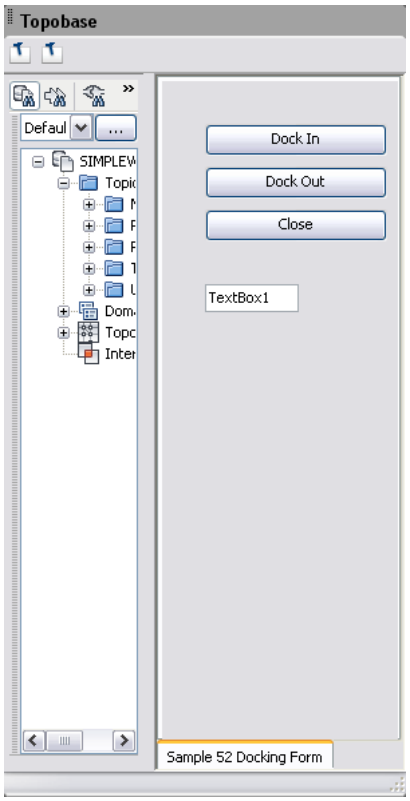
- Click Dock Out to show the form in a separate window.



- Click Close.
- Click Sample Dockin into Main Topobase Dialog.



Then click Dock In to dock the form into the Topobase task pane..



Sample 53 - Connection Tools

Purpose

This sample shows how to insert, select, and update rows in a table in the database. The code features the use of the following classes:

- `Topobase.Data.Tools.DataStructure`
- `Topobase.Data.Tools.RecordWriter`
- `Topobase.Data.Tools.RecordReader`
- `Topobase.Data.Tools.RecordBuffer`

- `Topobase.Data.Tools.ConnectionTools`
- `Topobase.Data.Tools.SqlExport`

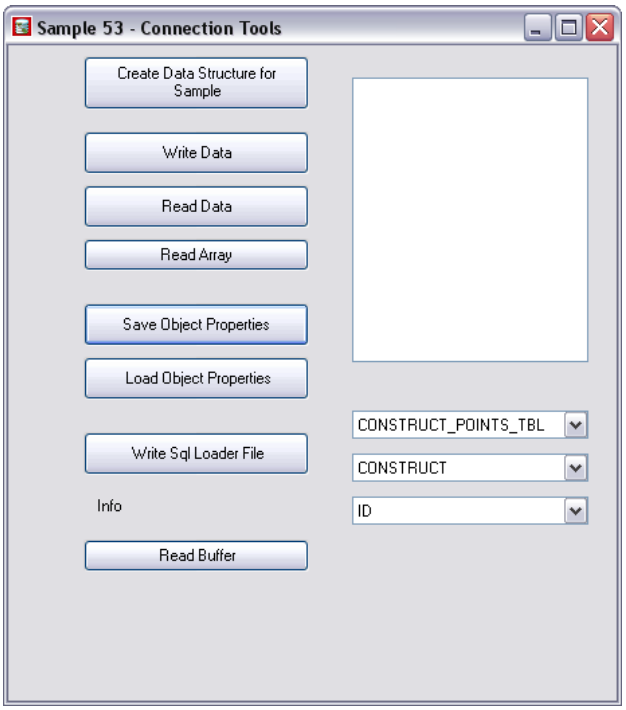
Procedure

To use this sample:

1. Build the sample. The plugin (*VBSample53.dll* or *CSSample53.dll*) will be placed in the Topobase Client *bin* directory. The *.tbp* file will also be copied to the Topobase Client *bin* directory.
2. Start Topobase and open the `TBSAMPLE` workspace.
3. Click the application toolbar button with the caption Sample 53 - Connection Tools. The new form is displayed.

   Note that if you have run the sample previously (you can use Step 6 to determine if the table exists) and want to start again from the beginning, you must remove the `sample_person` table from the database. In an Oracle SQL*Plus session enter the following commands:

   ```
   connect tbsample/avs
   drop table sample_person cascade constraints;
   ```



4 Click Create Data Structure For Sample. The handler for this button uses the `Topobase.Data.Tools.DataStructure` class. An information dialog box is displayed with the text Creating Structure. Click OK.

5 Verify that the table was created and determine its structure. In an Oracle SQL*Plus session enter the following commands:

```
connect tbsample/avs
select table_name from user_tables where table_name =
'SAMPLE_PERSON';
desc sample_person
```

6 Click Write Data. The handler for this button uses the `Topobase.Data.Tools.RecordWriter` class to set name/value pairs before performing the SQL command. An information dialog box is displayed with the text There are 0 values in SAMPLE_PERSON. Click OK.

7 A message box is displayed with a SQL statement for an insertion into SAMPLE_PERSON. Click OK. The inserted data is displayed in the list box; the source of the data is a select on SAMPLE_PERSON. This pattern of

events is repeated several times with some of the SQL statements being insertions and some being updates.

8. Click Read Data. The handler for this button uses the `Topobase.Data.Tools.RecordReader` class. Some data from each row in the SAMPLE_PERSON table is displayed in the list box.
9. Click Read Array. The handler for this button uses the `Topobase.Data.Tools.RecordBuffer` and the `Topobase.Data.Tools.ConnectionTools` classes. A message box is displayed with the text Gabi del la Blub has Size 0.58. Click OK.
10. Click Save Object Properties. The handler for this button uses the `Topobase.Data.Tools.RecordWriter` class to serialize properties extracted from an object before executing the insert SQL statement.
11. Click Load Object Properties. The handler for this button uses the `Topobase.Data.Tools.RecordReader` class to select a row from SAMPLE_PERSON and deserialize the properties to create an object. A message box is displayed with information extracted from the object. Click OK.
12. Click Write SQL Loader File. The handler for this button uses the `Topobase.Data.Tools.SqlExport` class to extract rows from SAMPLE_PERSON and write them to a file. A text editor window is displayed displaying the extracted information. Close the window.
13. Click Read Buffer. The handler for this button uses `Topobase.Data.Tools.RecordReader` class to select a row from SAMPLE_PERSON, uses the select results to create an object, displays two of the object's properties in a message box, changes one of the object properties that was displayed and then uses the `Topobase.Data.Tools.RecordWriter` class to extract the object properties and update the corresponding row in the SAMPLE_PERSON table. Finally, it displays the updated row in the list box.

Sample 55 - Drop Down Buttons

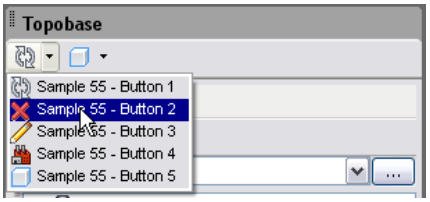
Purpose

This sample shows how to add drop down lists.

Procedure

To use this sample:

1. Build the sample. The plugin (*VBSample55.dll* or *CSSample55.dll*) will be placed in the Topobase Client *bin* directory. The *.tbp* file will also be copied to the Topobase Client *bin* directory.
2. Start Topobase. Two new icons appear on an application toolbar.



3. After you select one of the items from the menu, that item's icon is shown on the toolbar. The second button also demonstrates submenus.

Sample 56 - List Box

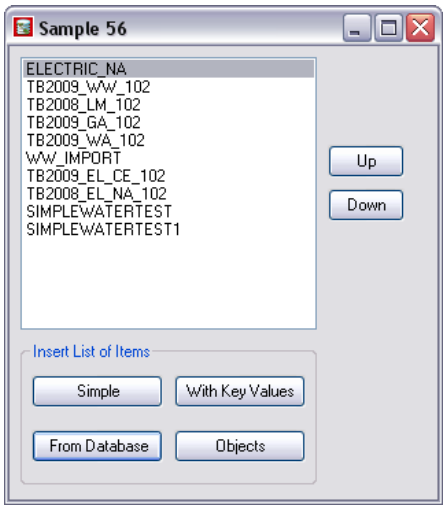
Purpose

This sample shows how to use the `Topobase.Forms.ListBox` control.

Procedure

To use this sample:

1. Build the sample. The plugin (*VBSample56.dll* or *CSSample56.dll*) will be placed in the Topobase Client *bin* directory. The *.tbp* file will also be copied to the Topobase Client *bin* directory.
2. Start Topobase.
3. Click the application toolbar button with the caption Sample 56 - Listbox. The plugin form is displayed.



Sample 57 - Special Menu Items

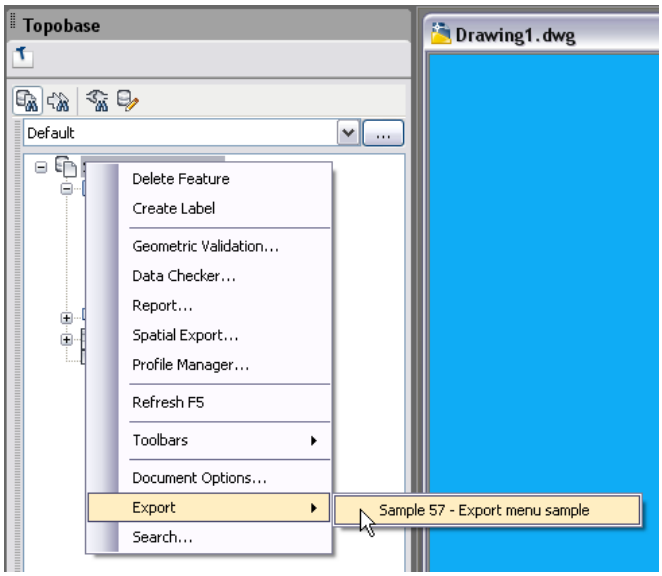
Purpose

This sample shows how to create special menu items for imports and exports. These differ from normal menu items because Autodesk Topobase shows all imports and exports in one list in the menu.

Procedure

To use this sample:

1. Build the sample. The plugin (*VBSample57.dll* or *CSSample57.dll*) will be placed in the Topobase Client *bin* directory. The *.tbp* file will also be copied to the Topobase Client *bin* directory.
2. Start Topobase and open any workspace.
3. Bring up the context menu of the base object in the Document Explorer.
4. Select the Export menu item. The sample has added a menu item called Sample 57 - Export menu sample



Sample 59 - Settings and Document Options

Purpose

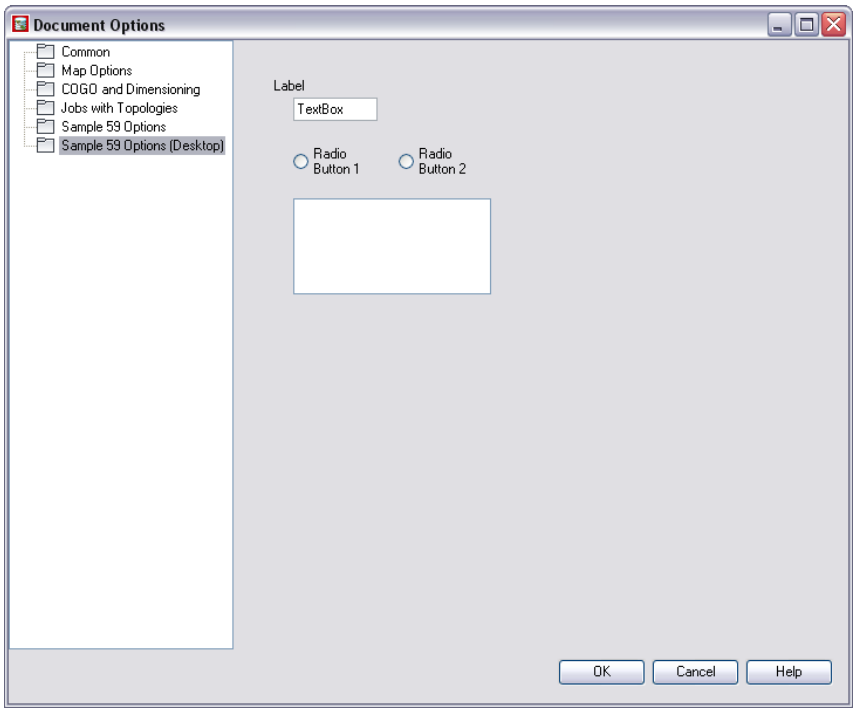
This sample shows how to create pages for the document option dialog box. It uses both the `Topobase.Forms.OptionPages.DocumentOptionPage` class (which works in both the desktop client and web client) and the Topobase.Forms.Desktop.DocumentOptionPage (which allows the use of Windows Forms elements, but cannot be used in the web client). It also demonstrates how to store settings for recording the options the user has selected.

Procedure

To use this sample:

1. Build the sample. The plugin (*VBSample59.dll* or *CSSample59.dll*) will be placed in the Topobase Client *bin* directory. The *.tbp* file will also be copied to the Topobase Client *bin* directory.
2. Start Topobase and open any workspace.

3 In the Settings ribbon, select Document Options. The sample's new Sample 59 Options and Sample 59 Options (Desktop) menu items appear in the tree on the left.



For a sample showing how to create *application* related options, see Sample 47 - Application Options (page 199).

Sample 61 - Dialog Box User Controls

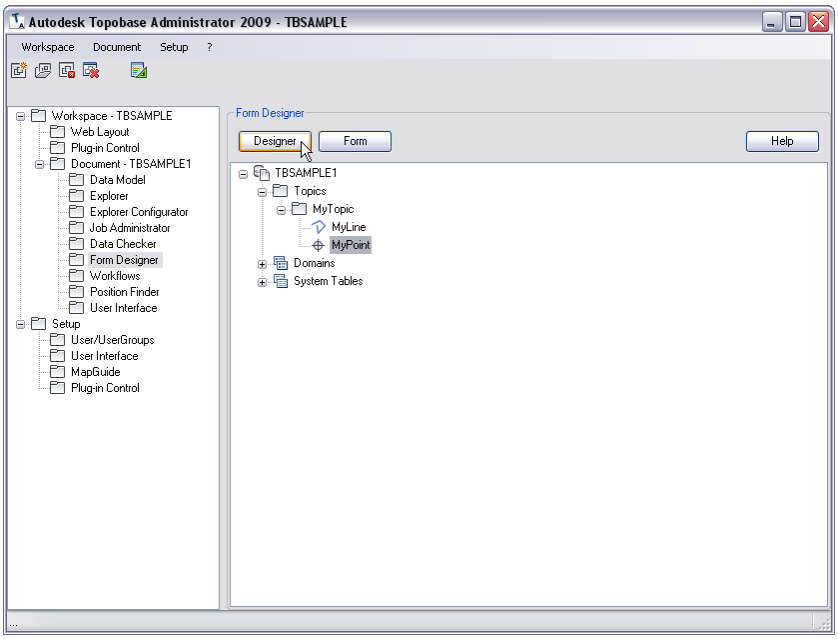
Purpose

This sample shows how to use controls in a generic dialog box. This sample uses the TBSAMPLE workspace created by the Sample 01 application.

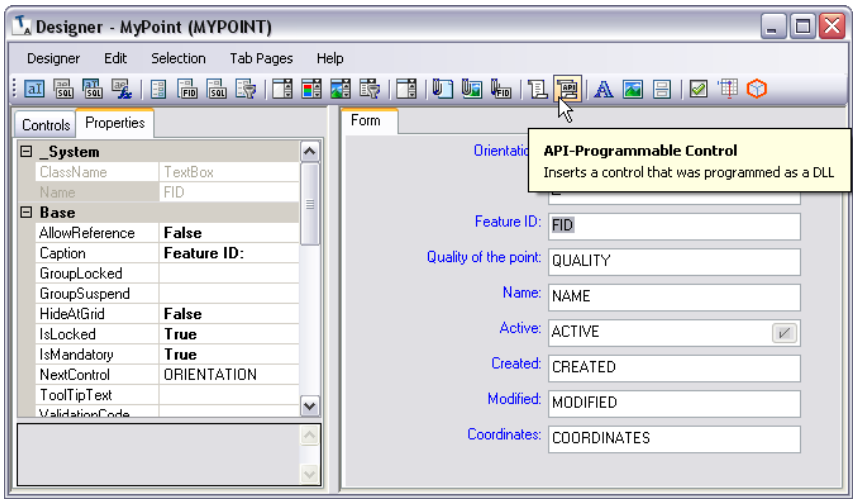
Procedure

To use this sample:

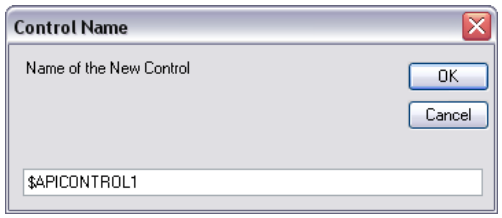
1. Build the sample. The plugin (*VBSample61.dll* or *CSSample61.dll*) will be placed in the Topobase Client */bin* directory. The *.tbp* file will also be copied to the Topobase Client *bin* directory.
2. Copy the *.dll* and *.tbp* files from the client */bin* directory to the administrator */bin* directory.
3. Start the Topobase Administrator and open the `TBSAMPLE` workspace.
4. In the Topobase Administrator, select the Form Designer in the left tree, and select MyPoint on the right.



5. Click Designer. The form designer opens.
6. Click the APIControl icon.



7 Name the new control `$APIControl1`. Click OK.

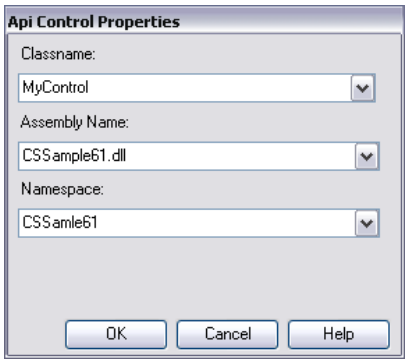


8 On the Properties tab, select the ControlClassName field and click the field's browse button.

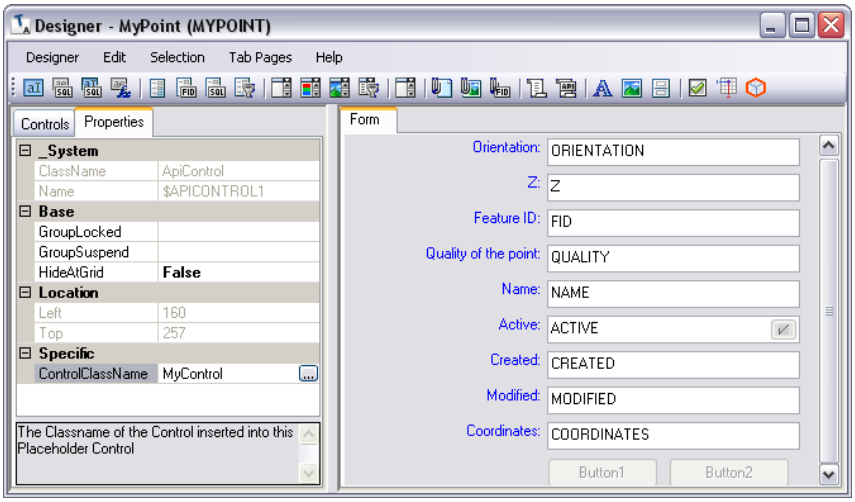
9 Set the Classname to MyControl.

10 Set the Assembly Name to CSSample61.dll if you are using C#, or to VBSample61.dll if you are using Visual Basic.

11 Set the Namespace to CSSample61 if you are using C#, or to VBSample61 if you are using Visual Basic. Click OK.



12 Two buttons are now shown where the control was placed.



13 Close the Designer form and Topobase Administrator.

14 Load the *.tbp* file (*VBSample61.dll* or *CSSample61.dll*) in the Topobase Client *bin* directory into a text editor. Modify the Name attribute of the DialogPlugIn element from `[Name="specified_table_name"]` to `[Name="MYPOINT"]`. Save the *.tbp* file.

15 Start Topobase Client and open the `TBSAMPLE` workspace.

16 In the Topobase task pane, right-click MyPoint and select Show Form. A new user control consisting of buttons labeled Button1 and Button2 is added to the form.

To restore the dialog box to its default state:

1. Start the Topobase Administrator.
2. Open the MyTopic dialog box again in the Forms Designer.
3. Select the control (the two buttons) on the Form tab.
4. Click Edit ➤ Delete Control.

Sample 62 - Control Test

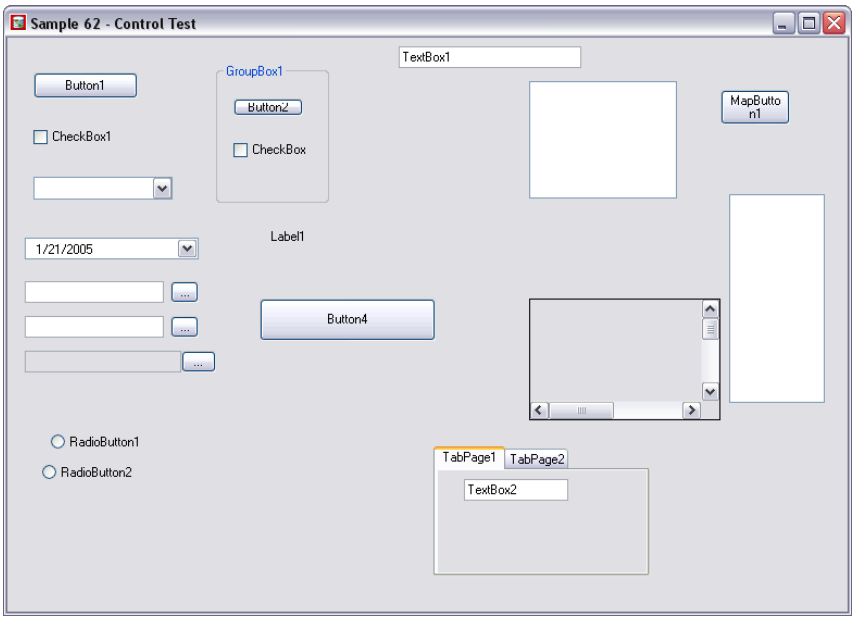
Purpose

This sample shows how to use controls in a generic dialog box.

Procedure

To use this sample:

1. Build the sample. The plugin (*VBSample62.dll* or *CSSample62.dll*) will be placed in the Topobase Client *bin* directory. The *.tbp* file will also be copied to the Topobase Client *bin* directory.
2. Start Topobase.
3. Click the application toolbar button with the caption Sample 62 - Controls. The new form is displayed that provides various functionality for buttons, text boxes, scrolling lists, and so on.



Sample 64 - Desktop Option Pages

Purpose

This sample shows how to create standard options dialog boxes using the `Topobase.Forms.Desktop.DocumentOptionPage` and `Topobase.Forms.Desktop.ApplicationOptionPage` classes. It also demonstrates the events that option pages have access to.

NOTE This sample only works in the desktop client, not in the web client.

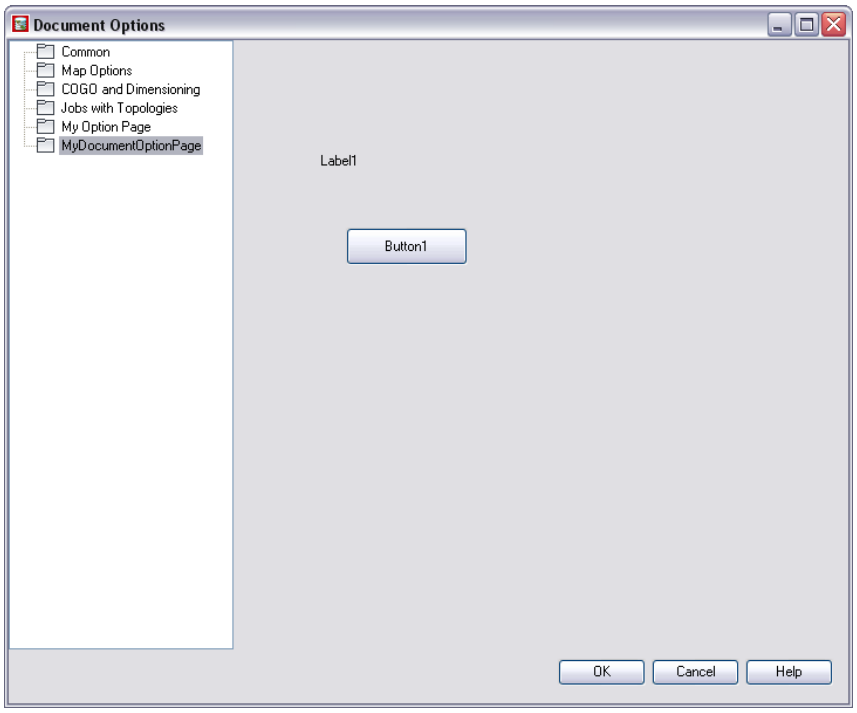
Procedure

To use this sample:

1. Build the sample. The plugin (*VBSample64.dll* or *CSSample64.dll*) will be placed in the Topobase Client *bin* directory. The *.tbp* file will also be copied to the Topobase Client *bin* directory.

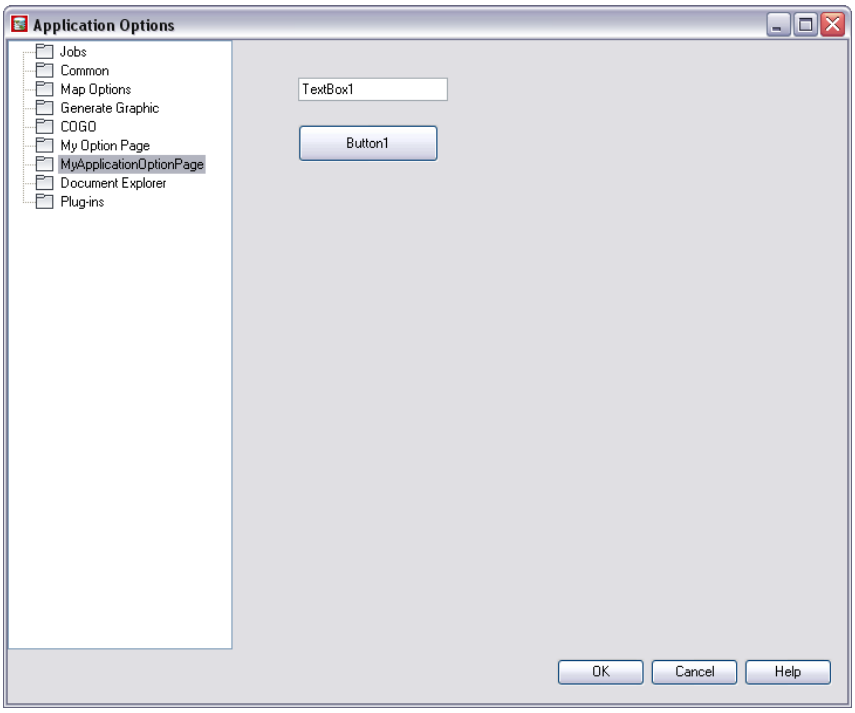
2 Start Topobase and open any workspace.

3 In the Settings ribbon, select Document Options.



The sample has added the MyDocumentOptionPage choice. You will see message boxes while performing actions with the Document Options dialog box as events in the sample option page fire.

4 In the Settings ribbon, selectApplication Options.



The sample has added the MyApplicationOptionPage choice. You will see message boxes while performing actions with the Application Options dialog box as events in the sample option page fire.

Sample 66 - Reports

Purpose

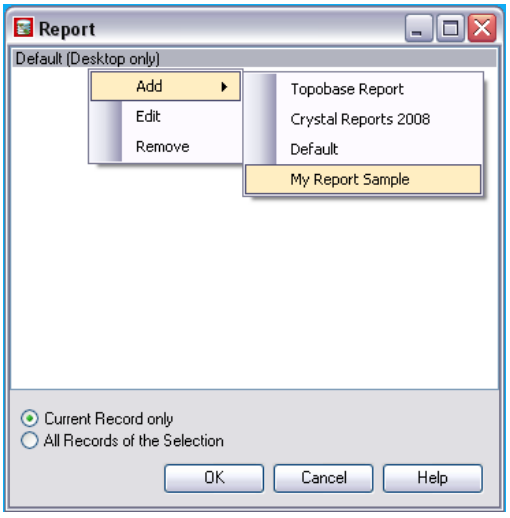
This sample shows how to create a connection to a report engine.

Procedure

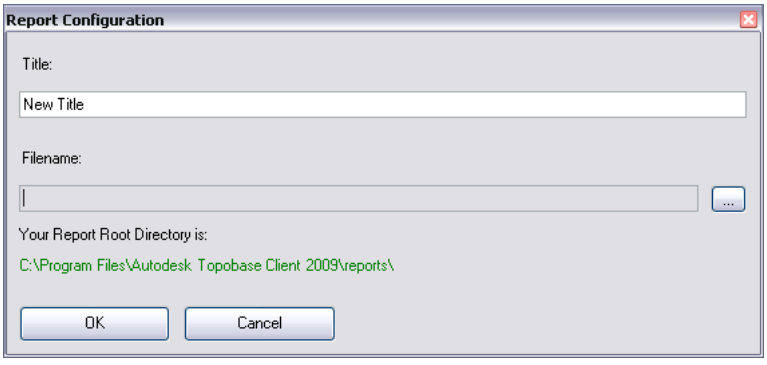
To use this sample:

1. Build the sample. The plugin (*VBSample66.dll* or *CSSample66.dll*) will be placed in the Topobase Client *bin* directory. The *.tbp* file will also be copied to the Topobase Client *bin* directory.
2. Start Topobase and open any workspace.
3. In the Topobase pane, right-click any feature and select Show Form.
4. In toolbar at the bottom of the feature form, click the Print icon.

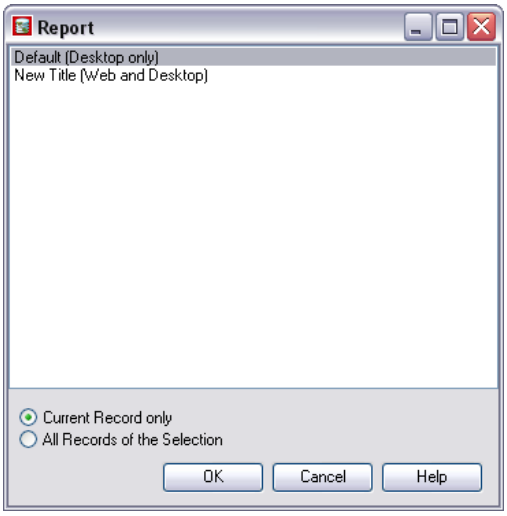
   The Report dialog box is displayed.
5. Right-click Default (Desktop only) and select My Report Sample.



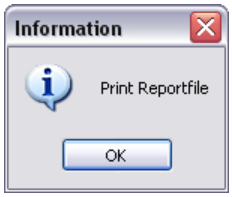
6. Enter a title for your report and click OK.



7 Select the report and click OK.



8 The event handler is triggered. In this case it just displays a dialog box.



Sample 67 - Dialog Box FlyIn

Purpose

This sample shows how to create a document FlyIn to customize a dialog box.

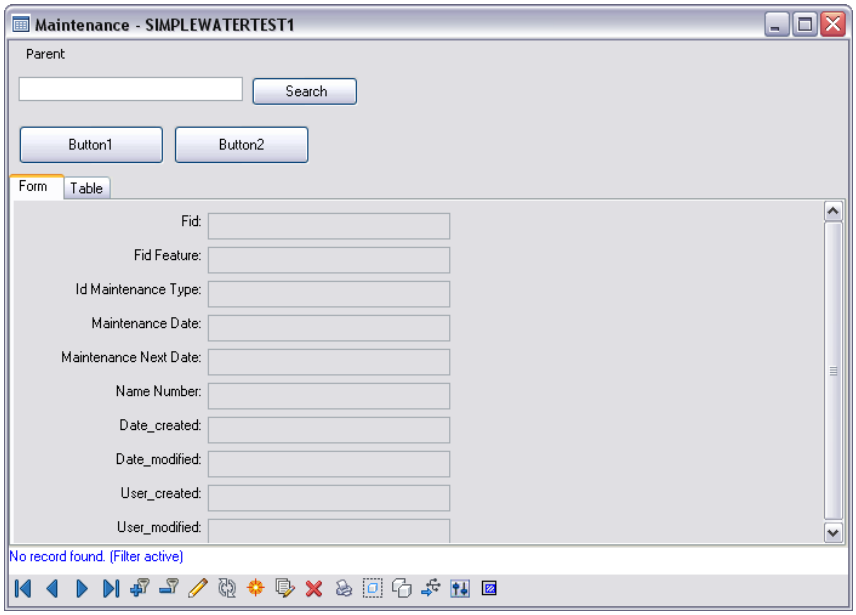
Procedure

To use this sample:

1 Build the sample. The plugin (*VBSample67.dll* or *CSSample67.dll*) will be placed in the Topobase Client *bin* directory. The *.tbp* file will also be copied to the Topobase Client *bin* directory.

2 Start Topobase and open any workspace.

3 In the Topobase pane, right-click any feature and select Show Form. The sample has added new controls to the dialog box.



To restore the dialog box to its default state remove the *.dll* and *.tbp* files from the Topobase *bin* directory, and restart Topobase.

Sample 68 - Option Page With Tree Nodes

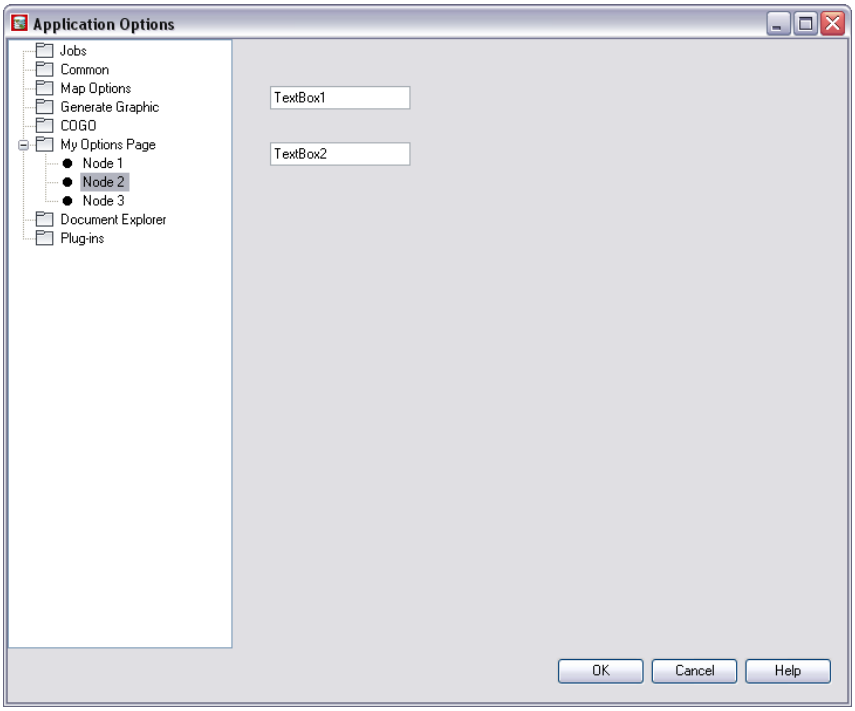
Purpose

This sample shows how to add an item with sub nodes to the document options tree.

Procedure

To use this sample:

1. Build the sample. The plugin (*VBSample68.dll* or *CSSample68.dll*) will be placed in the Topobase Client *bin* directory. The *.tbp* file will also be copied to the Topobase Client *bin* directory.
2. Start Topobase and open any workspace.
3. In the Settings ribbon, select Application Options. The sample has added a new item, My Options Page, to the treeview with three different nodes.



Sample 69 - FlyIn Container

Purpose

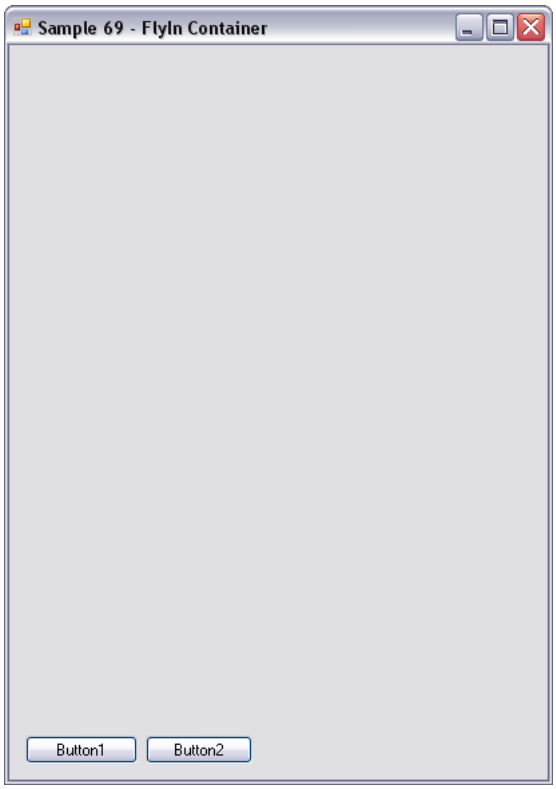
Sample 69 is an application that runs outside of Topobase. This sample shows how to create a standalone container form that can contain Topobase flyins based on the `Topobase.Forms.Desktop.FlyInMode.FlyIn` class. .

Procedure

To use this sample:

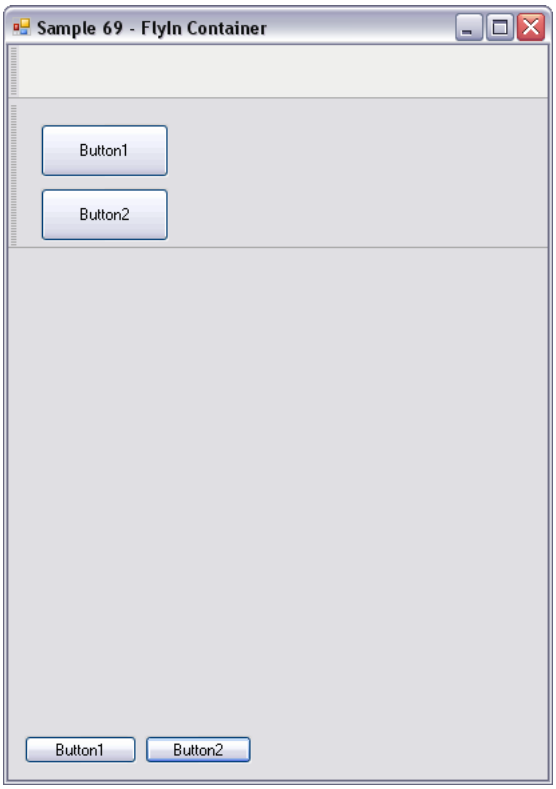
1. Run the program by either selecting Debug ➤ Start Debugging in Visual Studio or by directly running the sample's *.exe* file (by default, this is placed in the Topobase client *bin* directory). You do not need to run Topobase to run this sample.

   A dialog box with two buttons is displayed.



2 Click Button1. A new flyin containing an array of unlabeled toolbar buttons is created and docked within the form.

3 Click Button2. A new flyin containing two labeled buttons is created and docked within the form.



Sample 70 - Application Flyin

Purpose

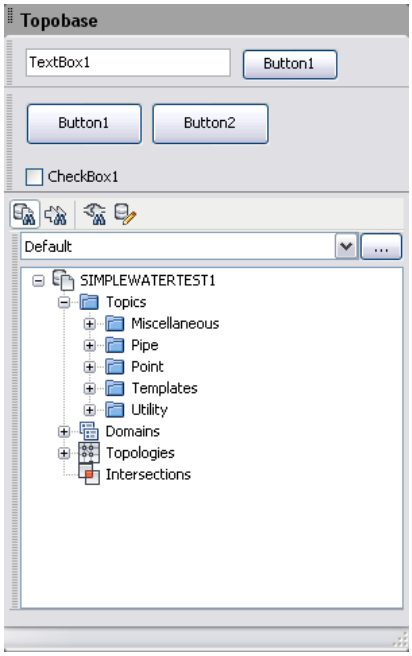
This sample shows how to add an application flyin to the Topobase task pane. Application flyins dock at the top of the Topobase task pane unlike document flyins, which dock at the bottom within the document tabs.

Procedure

To use this sample:

1 Build the sample. The flyin (*VBSample70.dll* or *CSSample70.dll*) will be placed in the Topobase Client *bin* directory. The *.tbp* file will also be copied to the Topobase Client *bin* directory.

2 Start Topobase. The sample adds the flyin to the top of the Topobase task pane. The flyin can be undocked by dragging it out of the Topobase task pane.



Sample 71 - Checked List Box

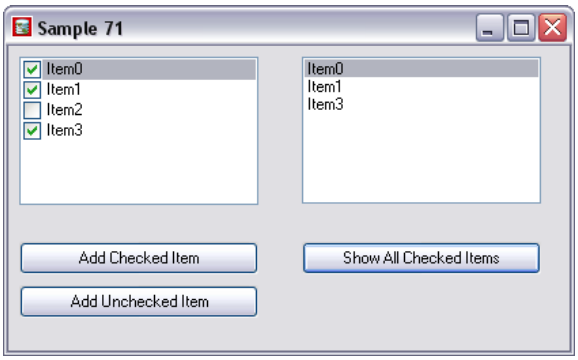
Purpose

This sample shows how to use the `Topobase.Forms.CheckedListBox` control.

Procedure

To use this sample:

1. Build the sample. The plugin (*VBSample71.dll* or *CSSample71.dll*) will be placed in the Topobase Client *bin* directory. The *.tbp* file will also be copied to the Topobase Client *bin* directory.
2. Start Topobase.
3. Click the application toolbar button with the caption Sample 71 - CheckedListBox. The new form is displayed that allows you to add checked or unchecked items to the left column, and show checked items in the right column.



Sample 72 - Application Toolbar

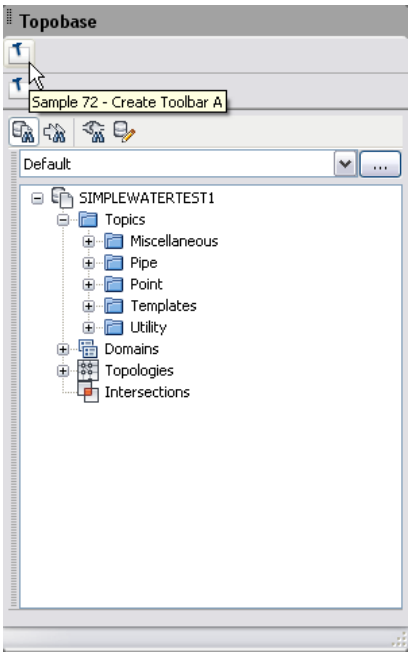
Purpose

This sample shows how to create custom application toolbars and add buttons to them.

Procedure

To use this sample:

1. Build the sample. The plugin (*VBSample72.dll* or *CSSample72.dll*) will be placed in the Topobase Client *bin* directory. The *.tbp* file will also be copied to the Topobase Client *bin* directory.
2. Start Topobase. The sample adds two new application toolbars to the Topobase task pane. Each toolbar contains one button.



3 If you click either button, a message box is displayed.

Sample 74 - List Box Multi Selection

Purpose

This sample shows how to use the `Topobase.Forms.Listbox` control to allow multiple selection.

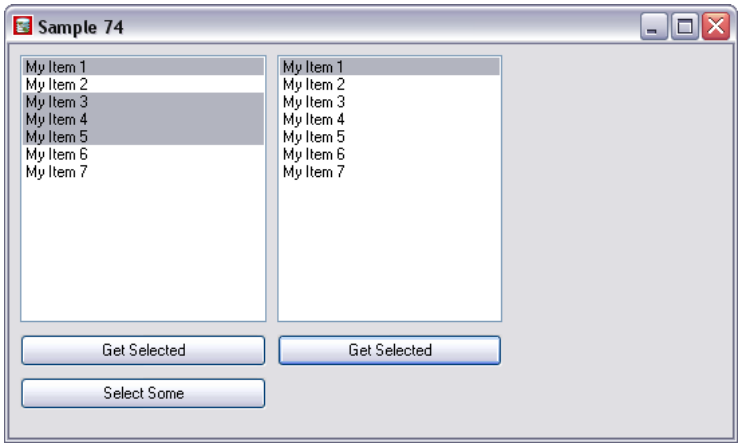
Procedure

To use this sample:

1 Build the sample. The plugin (*VBSample74.dll* or *CSSample74.dll*) will be placed in the Topobase Client *bin* directory. The *.tbp* file will also be copied to the Topobase Client *bin* directory.

2 Start Topobase.

3 Click the application toolbar button with the caption Sample 74 - ListBox with Multiselection. The plugin form is displayed. You can select multiple items by dragging through the list or by holding down the Control key and selecting multiple items. You can also select items programatically, as shown by the Select Some button.



Sample 75- SQL Export

Purpose

This sample shows how to export SQL queries. This sample demonstrates the use of the `Topobase.Data.Tools.SqlExport` class.

Procedure

To use this sample:

1 Build the sample. The plugin (*VBSample75.dll* or *CSSample75.dll*) will be placed in the Topobase Client *bin* directory. The *.tbp* file will also be copied to the Topobase Client *bin* directory.

2 Start Topobase and open any workspace.

3 Click the application toolbar button with the caption Sample 75 - SQL Export. A file named *test1.sql* is written to *C:\Program Files\Autodesk*

Topobase Client 2009\Temp. This file is then displayed in an instance of the *Notepad* application. This file contains the contents of the TB_TOPIC table. The source code contains many commented lines that can be used to modify the SQL export command.

4. Verify that the insert occurred. Do the following in an Oracle SQL*Plus session:
   1. `connect tbsample/avs`
   2. `select caption,id,name from tb_topic where id = 1;`

Sample 77 - Menu Control

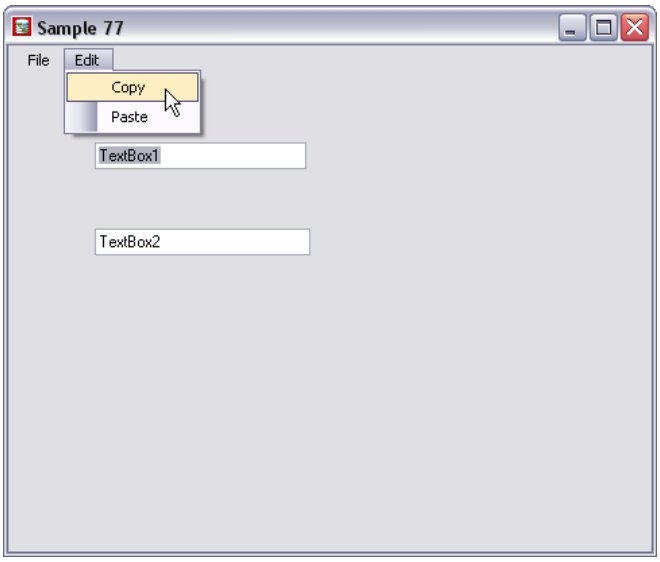
Purpose

This sample shows how to add a menu and menu items to a dialog box using the `Topobase.Forms.MenuControl` control.

Procedure

To use this sample:

1. Build the sample. The plugin (*VBSample77.dll* or *CSSample77.dll*) will be placed in the Topobase Client *bin* directory. The *.tbp* file will also be copied to the Topobase Client *bin* directory.
2. Start Topobase.
3. Click the application toolbar button with the caption Sample 77 - MenuControl. A plugin dialog box is displayed which demonstrates various menu functionality.



Sample 78 - Web Browser

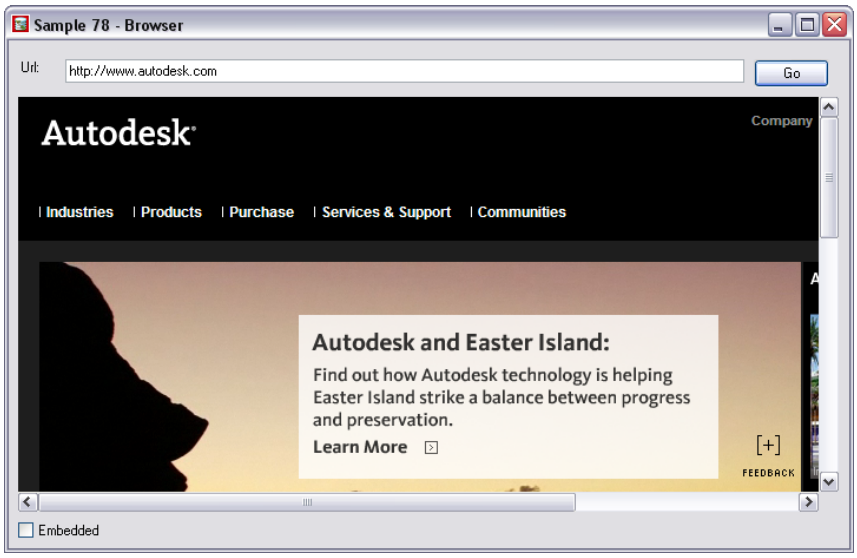
Purpose

This sample shows how to use a web browser control. Any document that can be opened by Internet Explorer can be opened by this control and shown in your form.

Procedure

To use this sample:

1. Build the sample. The plugin (*VBSample78.dll* or *CSSample78.dll*) will be placed in the Topobase Client *bin* directory. The *.tbp* file will also be copied to the Topobase Client *bin* directory.
2. Start Topobase.
3. Click the application toolbar button with the caption Sample 78 - Browser. The plugin form is displayed. A textbox at the top of the form allows you to type a web address which is displayed after clicking Go.



Sample 79 - Tab Control

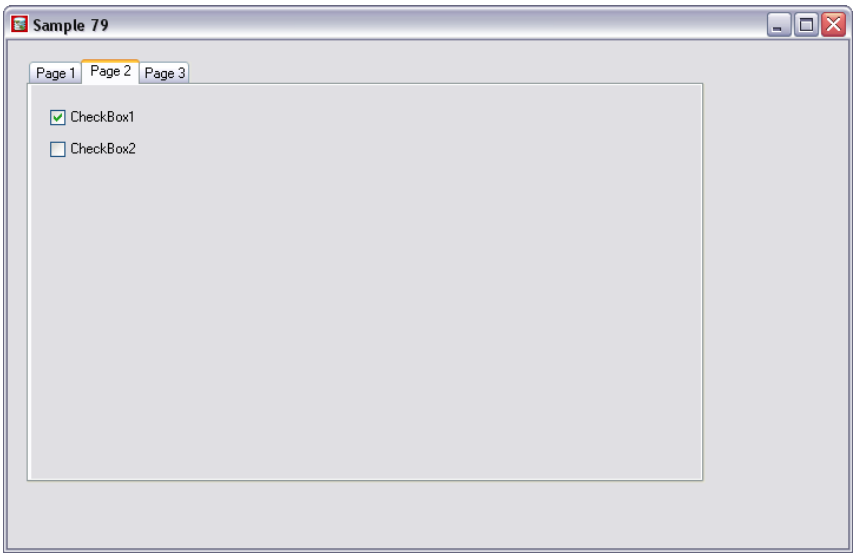
Purpose

This sample shows how to use a tab control and add tab pages.

Procedure

To use this sample:

1. Build the sample. The plugin (*VBSample79.dll* or *CSSample79.dll*) will be placed in the Topobase Client *bin* directory. The *.tbp* file will also be copied to the Topobase Client *bin* directory.
2. Start Topobase.
3. Click the application toolbar button with the caption Sample 79 - TabControl. A plugin form is displayed that demonstrates the tab functionality.



Sample 80 - Canvas Control

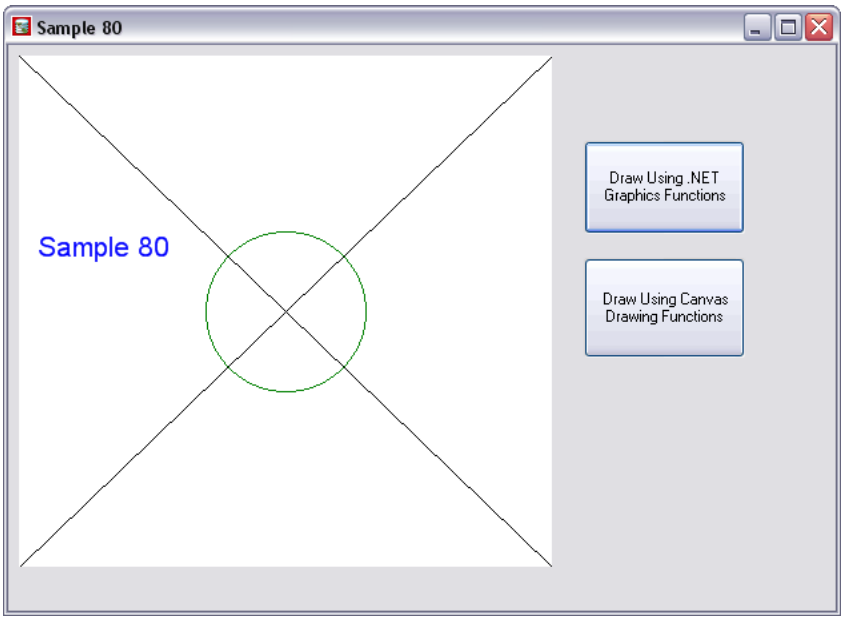
Purpose

This sample shows how to use a canvas control to draw lines, circles, etc.

Procedure

To use this sample:

1. Build the sample. The plugin (*VBSample80.dll* or *CSSample80.dll*) will be placed in the Topobase Client *bin* directory. The *.tbp* file will also be copied to the Topobase Client *bin* directory.
2. Start Topobase.
3. Click the application toolbar button with the caption Sample 80 - Canvas Control. The plugin form is displayed that demonstrates the canvas control functionality.



Sample 81 - Dialog Box Control Update

Purpose

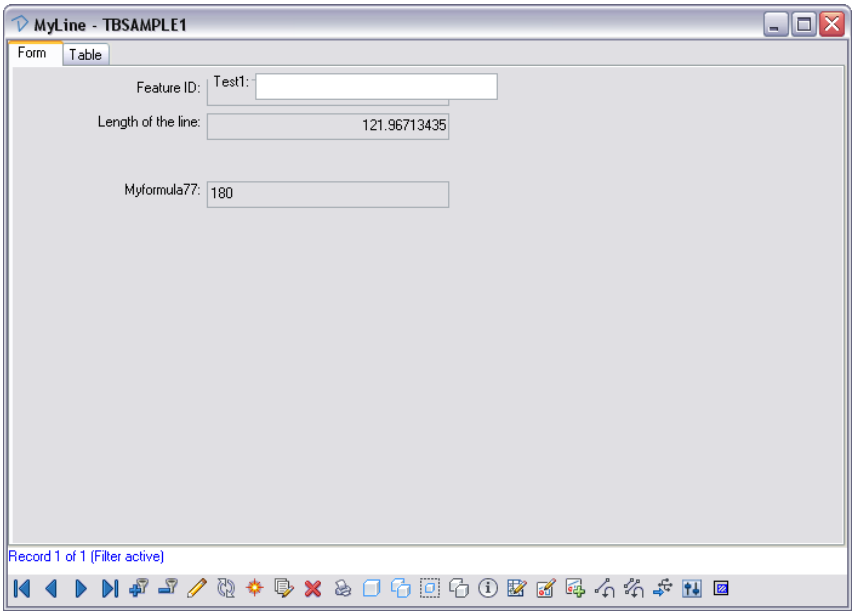
This sample shows how to add a control to a dialog box. This sample uses the TBSAMPLE workspace created by the Sample 01 application.

Procedure

To use this sample:

1. Build the sample. The plugin (*VBSample81.dll* or *CSSample81.dll*) will be placed in the Topobase Client *bin* directory. The *.tbp* file will also be copied to the Topobase Client *bin* directory.
2. Start Topobase and open the `TBSAMPLE` workspace.
3. In the Topobase pane, right-click MyLine and select Show Form. A new control is added to the form.

NOTE You cannot manage where the new control is placed through code. You can make room for the new control by moving the existing controls using the Form Designer in Topobase Administrator.



To restore the dialog box to its default state:

1. Remove the sample's *.dll* and *.tbp* files from the Topobase *bin* directory.
2. Start the Topobase Administrator and open the `TBSAMPLE` workspace.
3. Select the Form Designer in the left tree, and select the MyLine topic on the right.
4. Click Designer. The form designer opens.
5. Select the Test1 control and click Edit ➤ Delete Control.

The changes will take effect next time Autodesk Topobase is restarted.

Sample 83 - Remote Control

Purpose

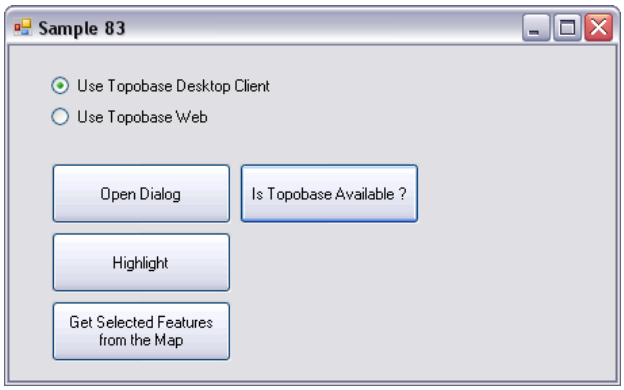
This sample shows how to control Topobase from another .Net application. It can be used in both the desktop client and web client. This sample features the use of the `Topobase.Forms.Remote.Client` class.

Procedure

This procedure assumes that you have added points to the TBSAMPLE workspace by either using the Sample 02 executable or digitizing point manually. To use this sample:

1. Modify Form1.cs as follows:
   - In the `ButtonOpenDialog_Click` method replace the first argument to the `OpenDialog` call with `"MYPOINT"`.
   - In the `ButtonHighlight_Click` method replace the argument to the `Add` call with a FID value from the MYPOINT table. (To determin a FID value, in an Oracle SQL*Plus session `connect tbsample/avs` and `select fid from mypoint;`.)
2. Build the sample. The executable (*VBSample83.exe* or *CSSample83.exe*) will be placed in the Topobase Client *bin* directory.
3. Ensure Topobase is closed.
4. Run the program by either selecting Debug ➤ Start Debugging in Visual Studio or by directly running the sample's *.exe* file in the Topobase *bin* directory.

   The sample's main form is displayed.



5 Click Is Available?. A message box is displayed with the text False. This indicates that Topobase is not available.

6 Start the Topobase client, and open the `TBSAMPLE` workspace.

NOTE If using the web client, start Topobase Web Client and switch the page to Feature Explore. If using the desktop client, start Topobase Client and open the workspace.

7 In sample form, click Is Available?. A message box is displayed with the text True. This indicates that Topobase is available.

8 Click Generate Graphic.

9 In sample Form1, click Open Dialog. The Topobase form for the MyPoint feature class is displayed, listing the points in the MYPOINT table.

10 In sample Form1, click Highlight. In the Topobase graphics pane the point whose FID you specified when you modified the `ButtonHighlight_Click` method is highlighted.

11 In sample Form1, click Get Selected Features From The Map. Select features in the Topobase graphics pane and press the Enter key when done. A message box with the FID number is displayed for each selected feature.

Sample 85 - Workflows

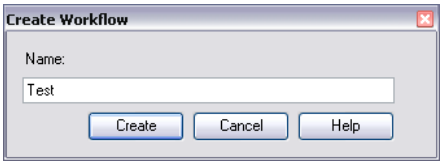
Purpose

This sample shows how to create simple workflows. This sample features the use of the `Topobase.Workflows.WorkflowScriptSupport` and `Topobase.Workflows.WorkflowPlugIn` classes. These workflows can be added to any workspace, including the TBSAMPLE workspace created by the Sample 1 application.

Procedure

To use this sample:

1. Build the sample. The plugin (*VBSample85.dll* or *CSSample85.dll*) will be placed in the Topobase Client *bin* directory. The *.tbp* file will also be copied to the Topobase Client *bin* directory.
2. Start Topobase and open a workspace. You may wish to use the TBSAMPLE workspace created by Sample 1.
3. In the Topobase task pane click Setup ➤ Administrator.
4. In the Topobase Administrator, select the folder labeled Workflows in the left pane.
5. Click Create.
6. In the Create Workflow dialog box enter a name for the workspace and click Create.



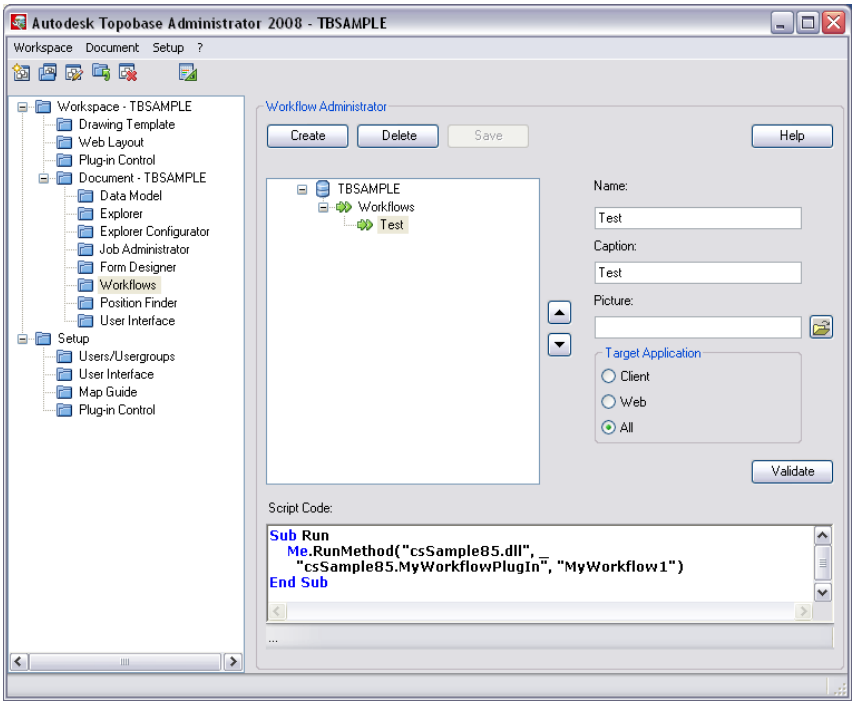
7. Select the workflow you just created, and copy this code into the Script Code edit field.
   - If you are using C#, use this:

```
Sub Run
 Me.RunMethod("csSample85.dll",
 "csSample85.MyWorkflowPlugIn", "MyWorkflow1")
```

```
End Sub
```

- If you are using Visual Basic, use this:

```
Sub Run
 Me.RunMethod("VBSample85.dll",
 "VBSample85.MyWorkflowPlugIn", "MyWorkflow1")
End Sub
```



8 Click Save.

9 Click Test. An information dialog box is displayed with the text Hello. Click OK. A dialog box labeled Topobase Workflows is displayed with the text The workflow has been executed.

10 Create and save another workflow.

- If you are using C#, use this for the code:

```
Sub Run
 Me.RunMethod("csSample85.dll",
 "csSample85.MyWorkflowPlugIn", "MyWorkflow2")
End Sub
```

- If you are using Visual Basic, use this for the code:

```
Sub Run
 Me.RunMethod("VBSample85.dll",
 "VBSample85.MyWorkflowPlugIn", "MyWorkflow2")
End Sub
```

11 (Re-)Start Topobase and open the workspace

12 Click Generate Graphic.

13 Click the Workflow Explorer icon in the Topobase task pane. The Workflow Explorer is displayed with three nodes representing the three workflows added by Sample 85.

14 Start the first workflow by highlighting the node and pressing Execute or by selecting Execute from the context menu of the node. This sample simply displays a message box and then end.

15 Start the second workflow. The Workflow Explorer contents are replaced with the text Please Digitize a Point.

16 Move the cursor into the graphics pane and click on a point. The Please Digitize a Point message is replaced with a text box labeled Coordinates, whose content is the digitized point values. Click OK. The text box is replaced with the contents of the Workflow Explorer.

Sample 86 - Update Plugin

Purpose

This sample shows how to update the data model of a workspace document. This sample uses the TBSAMPLE workspace created by the Sample 01 application.

Procedure

To use this sample:

1 Build the sample. The plugin (*VBSample86.dll* or *CSSample86.dll*) will be placed in the Topobase Client *bin* directory. The *.tbp* file will also be copied to the Topobase Client *bin* directory.

2 Start the Topobase Administrator and open the TBSAMPLE workspace.

NOTE Make sure you start the Topobase Administrator, not the Topobase Client.

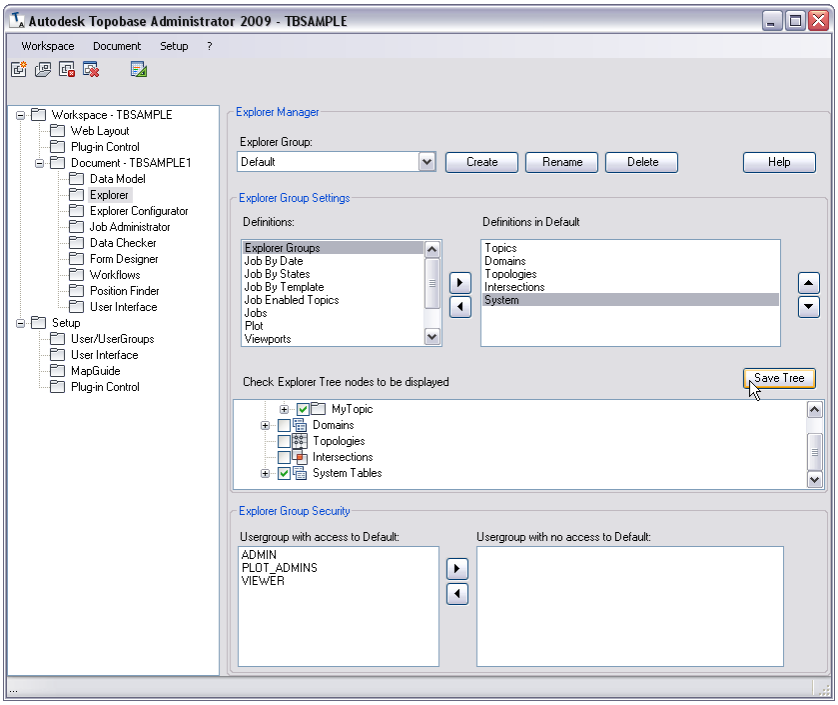
3 In the left tree, select Document - TBSAMPLE ➤ Explorer.

4 In the Definitions area, select System.

5 Click the right-arrow icon to move System to the Definitions in Default pane.

6 In the Check Explorer Tree Nodes To Be Displayed pane, check System Tables.

7 Click Save Tree.



8 Close Topobase Administrator.

9 Start Topobase Client and open the TBSAMPLE workspace.

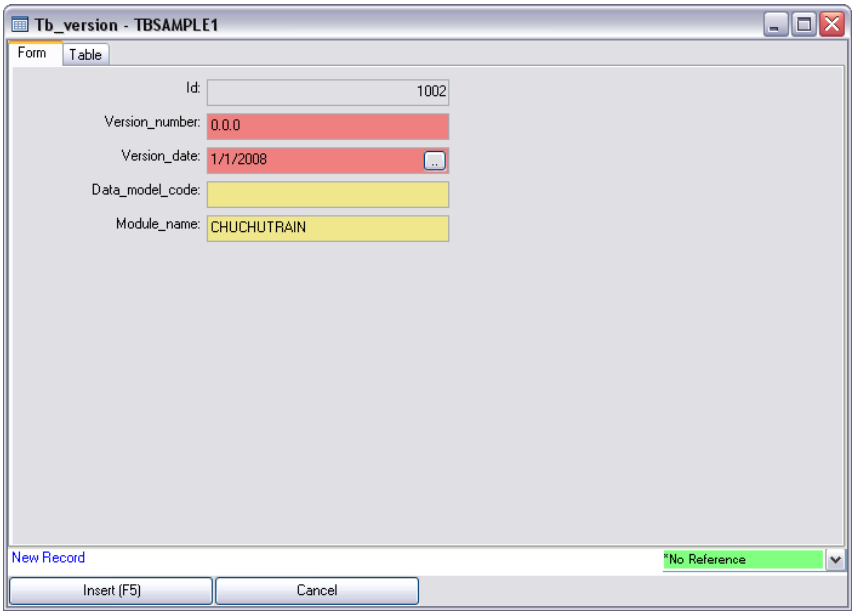
10 In the document explorer, expand the System Tables tree.

11 Under System Tables, right-click TB_VERSION and select Show Form from the context menu.

12 Select the New Record toolbar button and create a new record with these values:

- Version = 0.0.0
- Module Name = CHUCHUTRAIN

Set the Version_date to some date in the past.

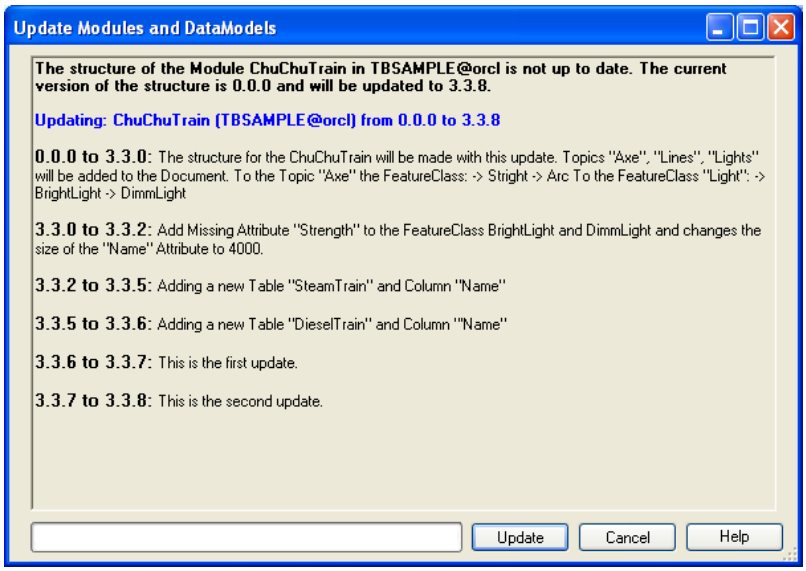


13 Click Insert (F5).

14 Close the form.

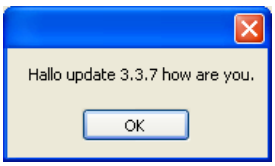
15 Close and re-open the workspace.

The Update Modules and Datamodules dialog box is displayed.



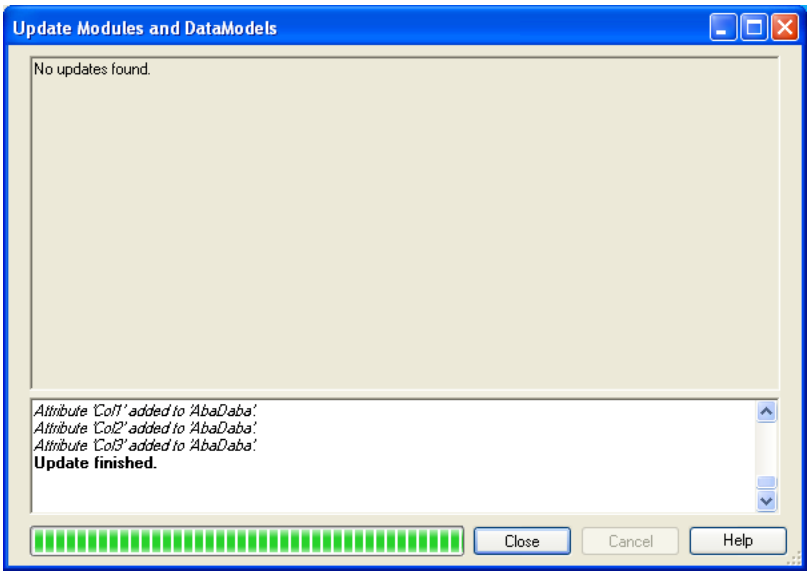
16 Click Update.

A dialog box is displayed during the update process.



17 Click OK.

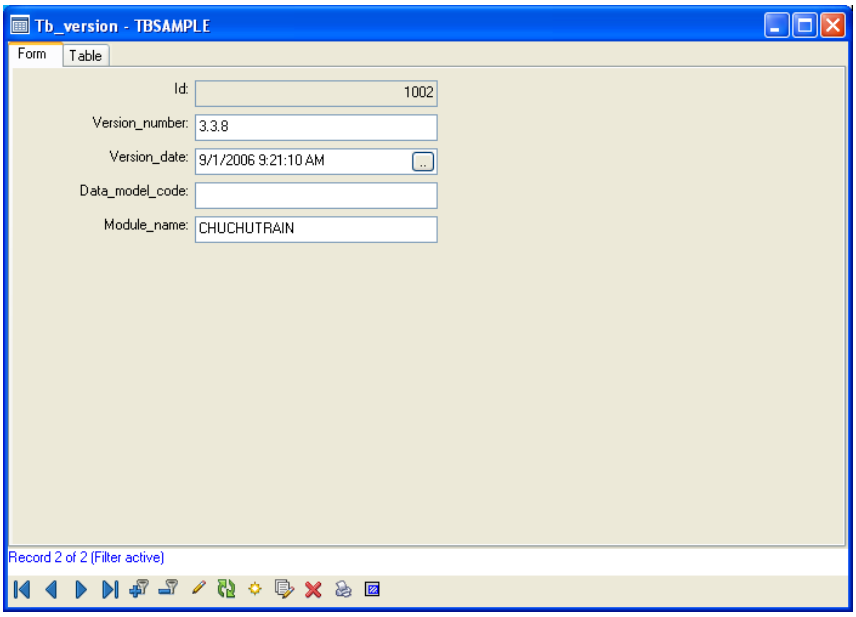
18 When the update is finished, the bottom area shows that the updates were applied successfully and the top area shows that there are no more updates to be applied.



19 Click Close.

The workspace then continues to open normally.

If you open the TB_VERSION form again, note that the version has been updated to 3.3.8 and the Version_date has been updated to the current date and time.



Sample 87 - Jobs

Purpose

This sample shows how to create a new job and how to change the state of a job. This sample uses the TBSAMPLE workspace created by the Sample 01 application.

Procedure

To use this sample:

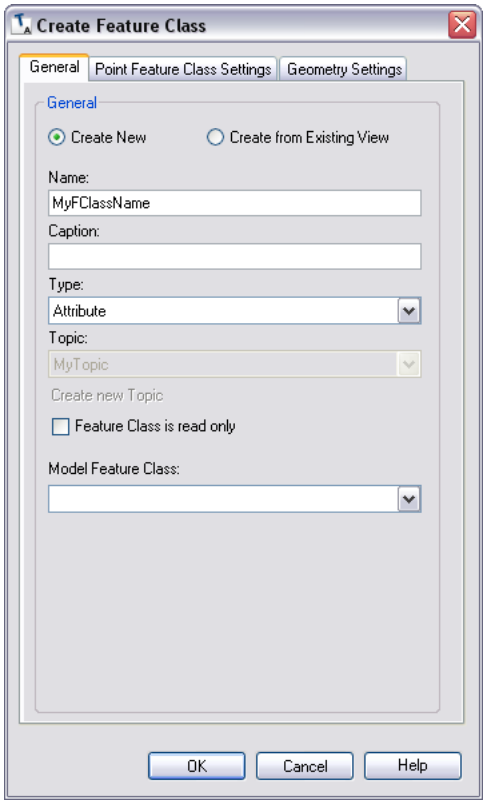
1. Build the sample. The plugin (*VBSample87.dll* or *CSSample87.dll*) will be placed in the Topobase Client *bin* directory. The *.tbp* file will also be copied to the Topobase Client *bin* directory.
2. Start the Topobase Administrator and open the `TBSAMPLE` workspace.

NOTE Make sure you start the Topobase Administrator, not the Topobase Client.

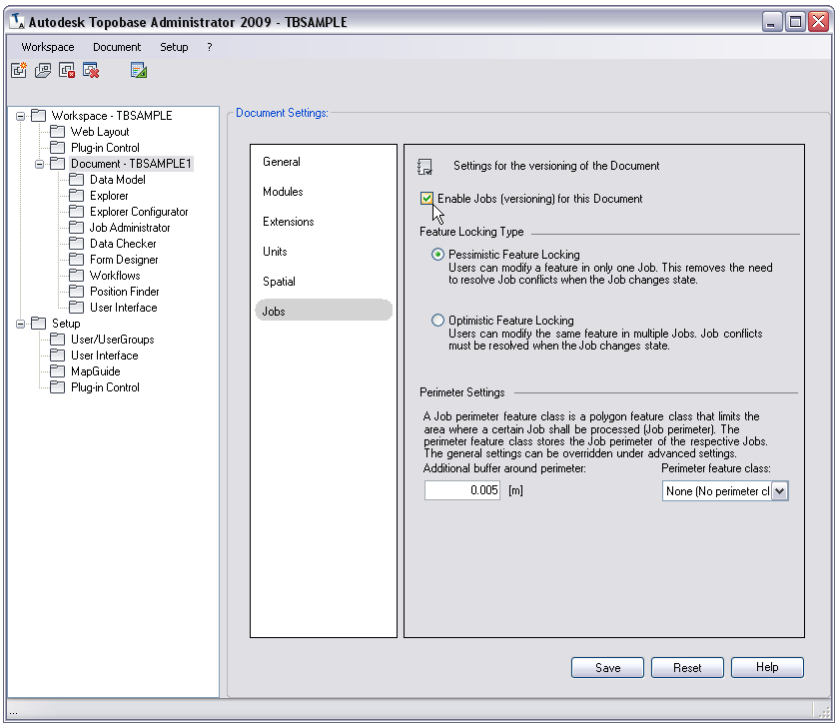
3 Create a new feature class called `MyFClassName` by first selecting Data Model in the left tree.

4 Right-click MyTopic on the right and select Create FeatureClass.

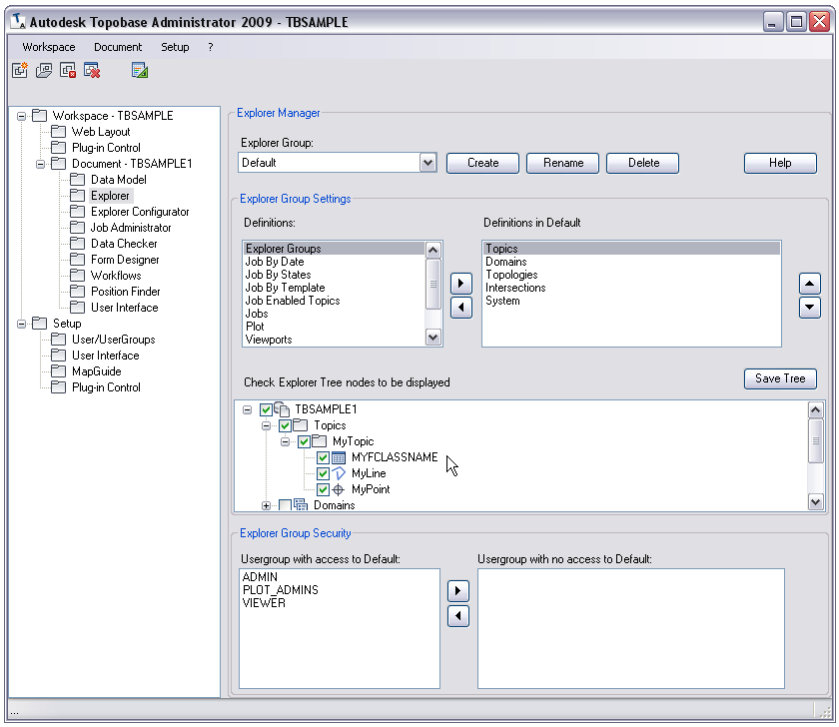
5 Enter the name of the new feature class MyFClassName and click OK.



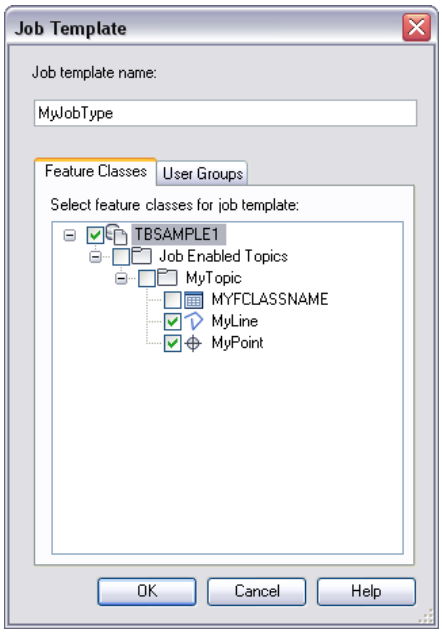
6 Enable jobs for this new feature class. Select Document - TBSAMPLE from the tree view. In the Document Settings page, select Jobs from the list. Check the check box labeled Enable Jobs (versioning) for this Document.



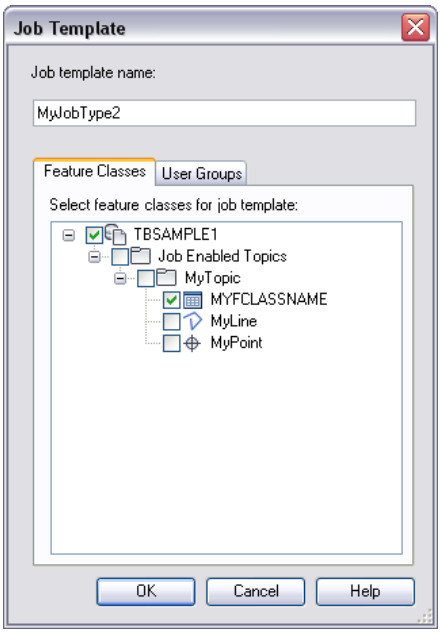
7 Set the new feature class to be shown in the Topobase document explorer. Select Explorer in the tree view. Check MyFClassName and click Save Tree.



8 Create a new job type. Select Job Administrator in the tree view. Expand the tree in the Job Templates and job transition passwords page. Click Create to bring up the Job Template dialog box. Use the template name MyJobType and check MyLine and MyPoint. Click Ok.



9 Create another job type with the name MyJobType2 and check MyFClassName.



10 Start Topobase and open the `TBSAMPLE` workspace.

11 Click the button labeled Sample 87 - Job Demo from the document toolbar.

The sample creates a new job, called My Job, that uses the feature class `MyFClassName`.

Sample 88 - Features

Purpose

This sample shows how to add a new feature class. This sample features the use of the `Topobase.Data.FeatureClasses` and `Topobase.Data.FeatureClass` classes. This sample uses the TBSAMPLE workspace created by the Sample 01 application.

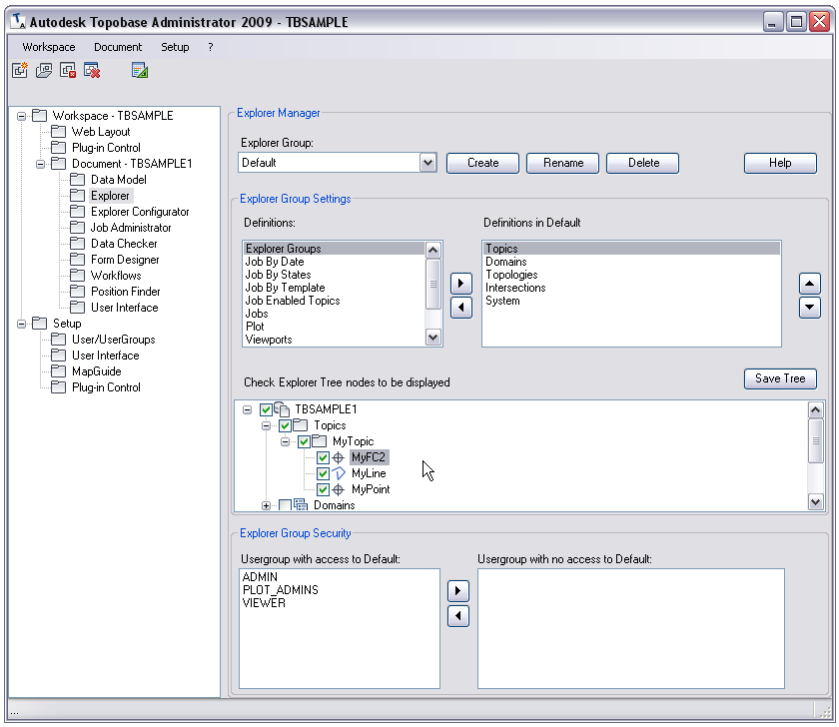
Procedure

Do the following:

1. Build the sample. The plugin (*VBSample88.dll* or *CSSample88.dll*) will be placed in the Topobase Client *bin* directory. The *.tbp* file will also be copied to the Topobase Client *bin* directory.
2. Start Topobase and open the `TBSAMPLE` workspace.
3. Select the document toolbar button labeled Sample 88 - Features.

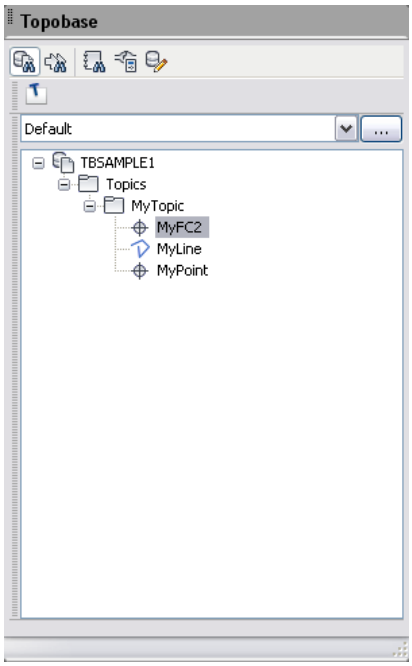
This action adds a feature class named MyFC2 to a topic named MyTopic in the TBSAMPLE workspace. However, the MyFC2 feature class is not yet displayed in the Document Explorer because it needs to be made visible using the Topobase Administrator first. To make the feature class visible, perform the following:

1. Start the Topobase Administrator and load the TBSAMPLE workspace.
2. In the tree view on the left-hand side, select Document - TBSAMPLE ➤ Explorer.
3. Check the MyFC2 checkbox.



4 Click Save Tree.

5 In the Topobase Client, close and re-open the TBSAMPLE workspace.



The new MyFC2 feature class is now visible under the MyTopic topic.

Sample 100 - Map Functions

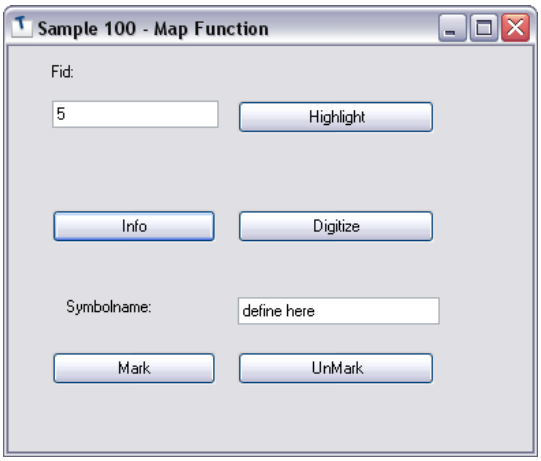
Purpose

This sample shows how to interact with Autodesk Map. This sample features the use of the `Topobase.Data.FeatureClass`, `Topobase.Data.FeatureList`, and `Topobase.Map.MapLogic` classes. This sample uses the TBSAMPLE workspace created by the Sample 01 application.

Procedure

To use this sample:

1. Build the sample. The plugin (*VBSample100.dll* or *CSSample100.dll*) will be placed in the Topobase Client *bin* directory. The *.tbp* file will also be copied to the Topobase Client *bin* directory.
2. Start Topobase and open the `TBSAMPLE` workspace
3. Click Generate Graphic.
4. Record some of the FIDs in the MyPoint and MyLine tables. You can do this by bringing up the forms for the MyPoint and MyLine feature classes. You can also use the line `connect tbsample/avs`, `select fid from mypoint;`, and `select fid from myline;` in an Oracle SQL*Plus session.
5. Select the application toolbar button labeled Sample 100 - Map Function. The sample's dialog box is displayed.



6. In the sample's dialog box type one of the MyPoint FIDs in the textbox labeled Fid:. Click Highlight. The point in the graphics pane with that FID is highlighted.
7. In the sample's dialog box type one of the MyLine FIDs in the textbox labeled Fid:. Click Highlight. The line in the graphics pane with that FID is highlighted.
8. In the sample's dialog box click Info. Move the cursor to the graphics pane and select some features by clicking, dragging, and clicking the cursor and the pressing the Enter key. An information dialog box is displayed with the text FID: <number>. Click OK. For every feature an information dialog box is displayed with its FID number.

9 In the sample's dialog box click Digitize. Move the cursor to the graphics pane. Click a point. Move the cursor around. You will note that the text attached to the cursor changes. Click again. An information dialog box is displayed with the FID of a point that has been added to the MyPoint table in the Oracle database. Click OK. In an Oracle SQL*Plus session `connect tbsample/avs` and `select fid,geom from mypoint;`. You will note that sometimes the coordinates of the point added are integral and are the same as the point you first clicked and sometimes the coordinates have fractional parts.

Sample 102 - Picture Combobox

Purpose

This sample shows how to use the picture combobox control.

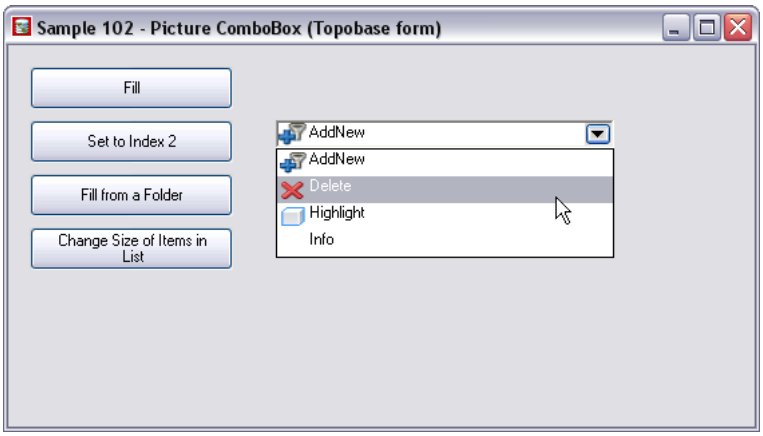
Procedure

To use this sample:

1 Build the sample. The plugin (*VBSample102.dll* or *CSSample102.dll*) will be placed in the Topobase Client *bin* directory. The *.tbp* file will also be copied to the Topobase Client *bin* directory.

2 Start Topobase.

3 Click the application toolbar button with the caption Sample 102 - Picture ComboBox. The sample's dialog box is displayed.



4 Click Fill. The dropdown box is populated with four entries: AddNew, Delete, Highlight, and Info in that order.

5 Click Set To Index 2. The dropdown box shows the Highlight entry. An information dialog box is displayed with the text Highlight. Click OK.

6 Click Fill From A Folder. The dropdown box is populated with the list of files in the *C:\Program Files\Autodesk Topobase Client 2009\Pics* folder.

7 Click Change Size Of Items In List. Click the control on the dropdown box. The graphic entries in the list have increased in size.

Sample 103 - Call Class Via Batch File

Purpose

This sample shows how to use a batch file to get information about feature classes.

Procedure

To use this sample:

1 Build the sample. The batch client (*VBSample103.dll* or *CSSample103.dll*) will be placed in the Topobase Client *bin* directory.

NOTE Since this DLL is called by *TBBatch.exe*, and not by the main Topobase application, there is no *.tbp* file for this sample.

2 Copy the *Config103.xml* file that is supplied with the example to the *bin* directory.

3 If necessary, edit the passwords and any other parameters in *Config103.xml*.

```
<?xml version="1.0" encoding="utf-8"?>
<Batch>
 <Assembly
 AssemblyName="CSSample103.dll"
 Namespace="CSSample103"
 ClassName="Export"
 />
 <Connection
 Username="TBSAMPLE"
 PassWord="AVS"
 Datasource="ORCL"
 tbsysUsername="TBSYS"
 tbsysPassword="TBSYS"
 tbsysDataSource="ORCL"
 />
 <Data
 OutputFilename="C:\Test.txt"
 SeparatorSign=";"
 >
 <FeatureClassNames>
 <FeatureClassName Value="MyPoint"/>
 <FeatureClassName Value="MyLine"/>
 </FeatureClassNames>
 <Colors>
 <Color Key="A" Value="Red"/>
 <Color Key="B" Value="Green"/>
 <Color Key="C" Value="Blue"/>
 </Colors>
 </Data>
</Batch>
```

4 At the DOS command line, enter the following commands:

```
cd /d "C:\Program Files\Autodesk Topobase Client 2009\Bin"
TBBatch.exe ConfigFile=Config103.xml
```

The file specified in the configuration file, *C:\test.txt*, will now read:

```
FeatureClass:MyPoint
FeatureClass:MyLine
A - Red
B - Green
C - Blue
```

Sample 104 - Script Engine

Purpose

This sample shows how to run short scripts, such as VB.NET scripts, at runtime.

NOTE The code that defines the scripts does not need to be written in Visual Basic. It can be written in C#. For example, see the *MyApplication.cs* file for this sample.

Procedure

To use this sample:

1. Build the sample. The plugin (*VBSample104.dll* or *CSSample104.dll*) will be placed in the Topobase Client *bin* directory. The *.tbp* file will also be copied to the Topobase Client *bin* directory.
2. Start Topobase.
3. Click the application toolbar button with the caption Sample 104 - Script Engine. The scripts defined within the sample are executed and a series of message boxes are displayed with the results.

Sample 105 - Create Topobase User

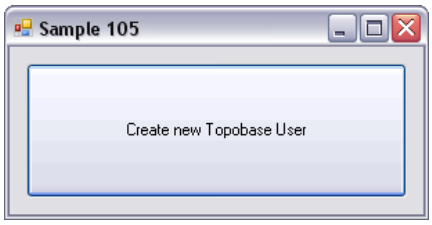
Purpose

This sample shows how to create a new Topobase user. This sample uses the TBSAMPLE workspace created by the Sample 01 application.

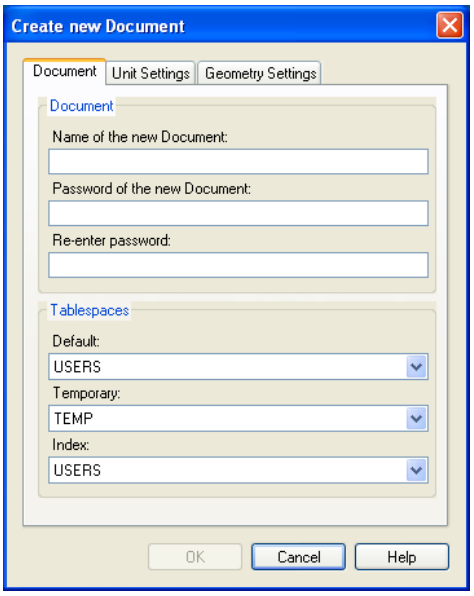
Procedure

To use this sample:

1. Build the sample. The plugin (*VBSample105.dll* or *CSSample105.dll*) will be placed in the Topobase Client *bin* directory. The *.tbp* file will also be copied to the Topobase Client *bin* directory.
2. Start Topobase and open the `TBSAMPLE` workspace.
3. Click the application toolbar button labeled Sample 105 - Create User. The sample's dialog box is displayed.



4. Click the button. The Create New Document dialog box is displayed.



5 You can then fill in the parameters to create the new document.

Sample 106 - Unit Support

Purpose

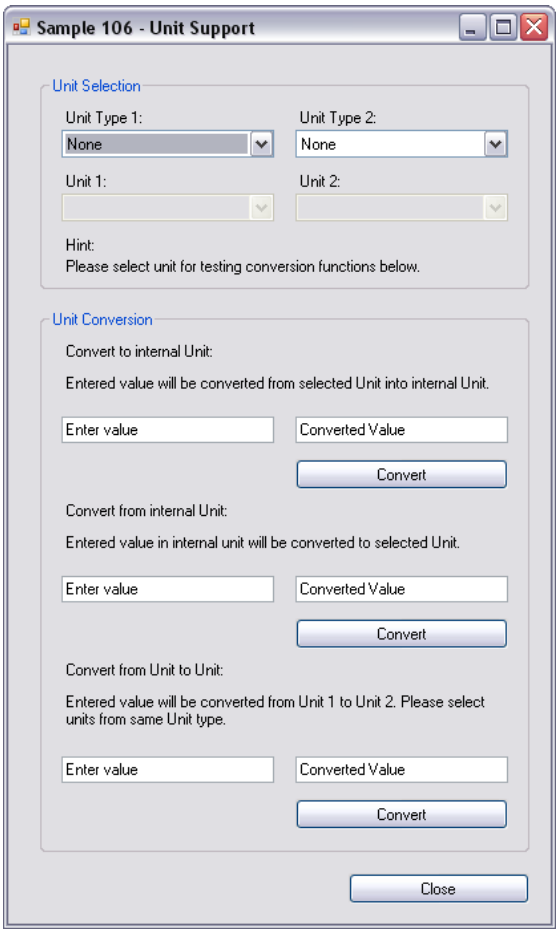
This sample shows how to use the unit framework. This contains all the supported unit types with corresponding units and conversion functions.

Procedure

To use this sample:

1 Run the program by either selecting Debug ➤ Start Debugging in Visual Studio or by directly running the sample's *.exe* file (by default, this is placed in the Topobase client *bin* directory). You do not need to run Topobase to run this sample.

The following form is displayed.



2 You can now use the form to convert various unit types and values.

Sample 111 - Feature Explorer

Purpose

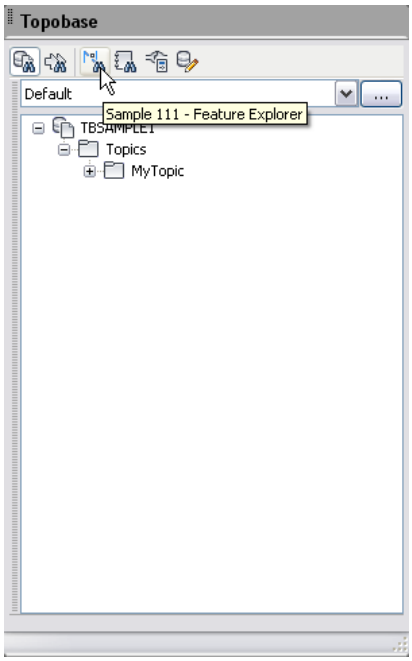
This sample shows how to create a Feature Explorer FlyIn, and it works in both Client and Web mode. This sample uses the TBSAMPLE workspace created by the Sample 01 application. It features the use of the following classes:

- `Topobase.Forms.Desktop.DocumentFlyIn`
- `Topobase.Forms.UserControls.FeatureExplorer.FeatureExplorerData`
- `Topobase.Forms.UserControls.FeatureExplorer.AbstractDesktopFeatureExplorerController`
- `Topobase.Forms.UserControls.FeatureExplorer.Action`
- `Topobase.Forms.UserControls.FeatureExplorer.Category`
- `Topobase.Data.FeatureClass`
- `Topobase.Data.Feature`
- `Topobase.Graphic.LineString`

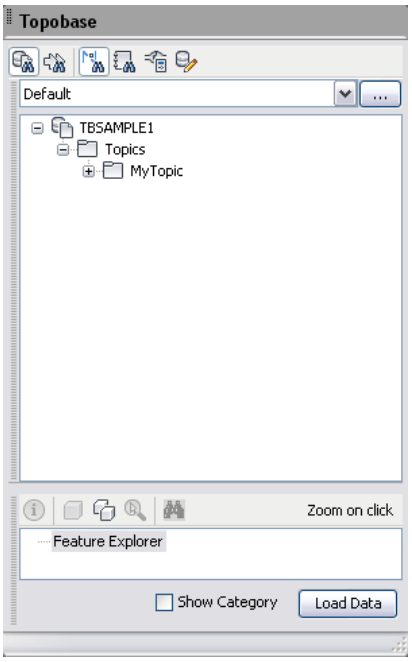
Procedure

To use this sample:

1. Build the sample. The plugin (*VBSample111.dll* or *CSSample111.dll*) will be placed in the Topobase Client *bin* directory. The *.tbp* file will also be copied to the Topobase Client *bin* directory.
2. Start Topobase and open the `TBSAMPLE` workspace. The main toolbar of the Topobase task pane has an additional button labeled Sample 111 - Feature Explorer next to the buttons for the Document Explorer and Workflow Explorer.
3. Click Generate Graphic.
4. Click the explorer toolbar button labeled Sample 111 - Feature Explorer. This will display the Feature Explorer flyin at the bottom of the Topobase task pane.



5 Click Load Data in the Feature Explorer flyin. Two subnodes labeled MyLine and MyPoint are displayed under the Feature Explorer node. Expand these nodes to display the FIDs for the MyPoint and MyLine features.



Sample 112 - Client Side Feature Rules

Purpose

This sample shows how to create a client side feature rule. This sample uses the TBSAMPLE workspace created by the Sample 01 application. It features the use of the `Topobase.Data.FeatureRules.FeatureRulePlugIn` and `Topobase.Data.FeatureRules.FeatureRuleDef` classes.

Procedure

To use this sample:

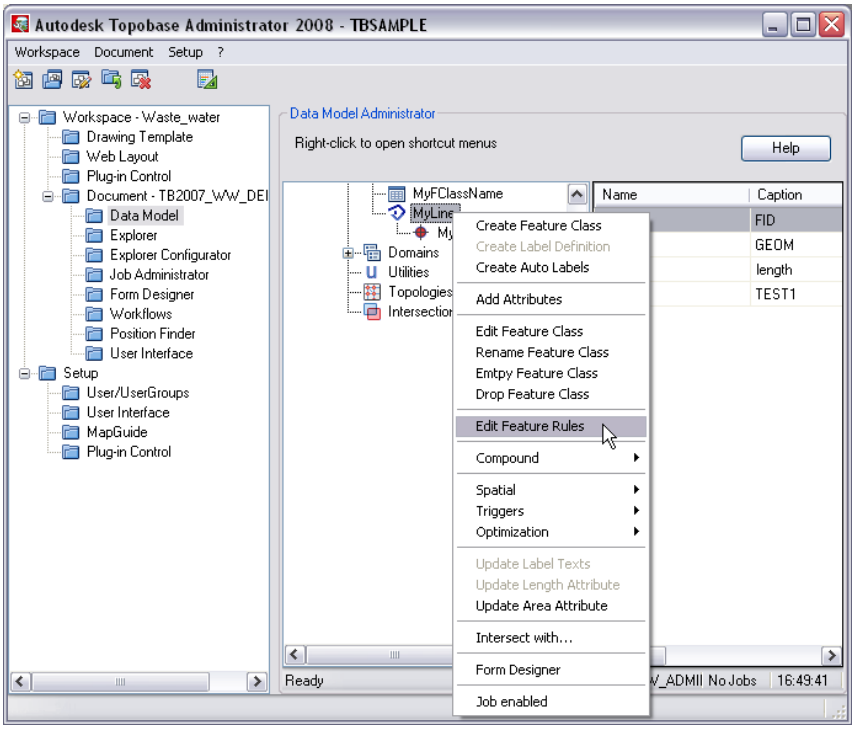
1. Build the sample. The plugin (*VBSample112.dll* or *CSSample112.dll*) will be placed in the Topobase Client *bin* directory. The *.tbp* file will also be copied to the Topobase Client *bin* directory.

2 Start the Topobase Administrator and open the `TBSAMPLE` workspace.

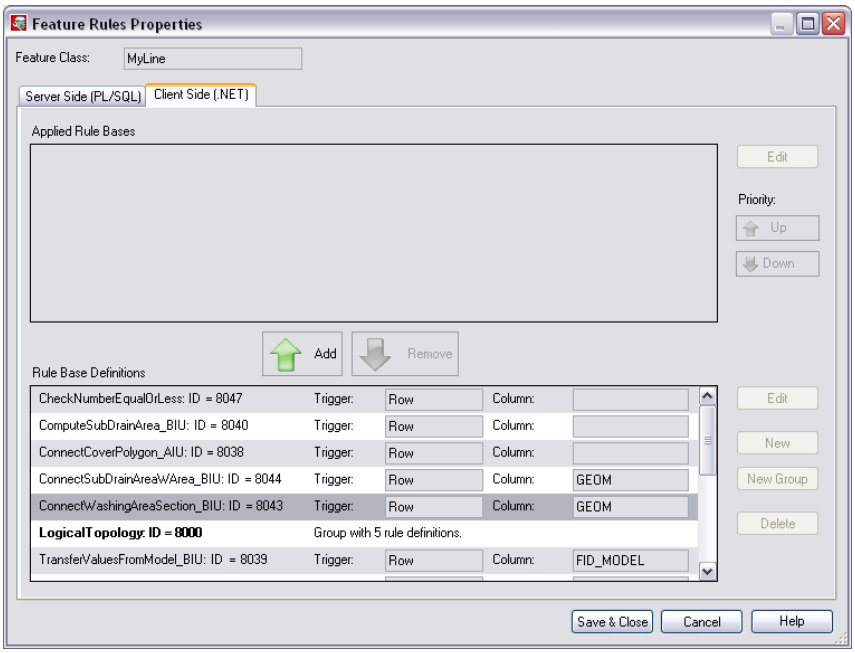
NOTE Make sure you start the Topobase Administrator, not the Topobase Client.

3 Select Data Model in the left tree.

4 Right-click a feature and select Edit Feature Rules.



The Feature Rules Properties dialog box opens. Select the Client Side (.NET) tab.



Sample 116 - Advanced Workflows

Purpose

This sample shows how to create a more complicated workflow than those found in Sample 85, the first workflow sample. This workflow digitizes a new pipe between a point and the selected pipe main.

This sample requires a workspace using a compatible water utility data model.

Installing the Workflow

To install the workflow:

1. Build the sample. The plugin (*VBSample116.dll* or *CSSample116.dll*) will be placed in the Topobase Client *bin* directory. The *.tbp* file will also be copied to the Topobase Client *bin* directory.

2 Start the Topobase Administrator and open the water utility workspace.

NOTE Make sure you start the Topobase Administrator, not the Topobase Client.

3 In the Topobase Administrator, select the folder labeled Workflows in the left pane.

4 Click Create.

5 In the Create Workflow dialog box enter a name for the workspace (such as "Sample116: House Connection creation") and click Create.

6 Select the workflow you just created.

7 Copy the following code into the Script Code edit field:

- If you are using C#:

```
Sub Run
 Me.RunMethod("csSample116.dll", _
 "csSample116.MyWorkflowPlugIn", "HouseConnectionCreation")
End Sub
```

- If you are using Visual Basic:

```
Sub Run
 Me.RunMethod("VBSample116.dll", _
 "VBSample116.MyWorkflowPlugIn", "HouseConnectionCreation")
End Sub
```

8 Click Validate to make sure you typed in the code correctly.

9 Select All under "Target Application

10 Click Save and exit Topobase Administrator.

Procedure

To use this sample workflow:

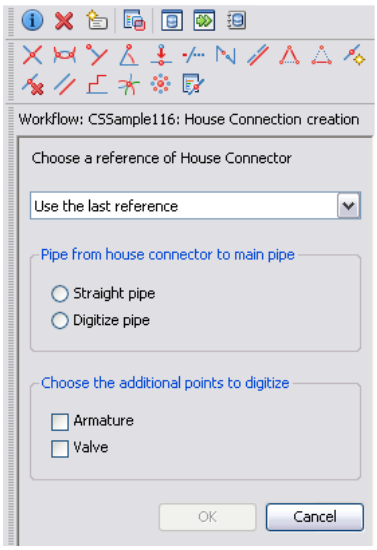
1 Start Topobase Client and open the TBSAMPLE workspace.

2 Click Generate Graphic.

3 Click the Workflow Explorer icon.

4 Right-click the workflow node labeled with the name of the workflow that you created and click Execute. The workflow will start and the

Workflow Explorer contents are replaced with the following workflow form:



Sample 117 - Simple Water Module

Purpose

Sample 117 is a simple vertical application module that contains the minimum requirements for a functional module. It contains the following features:

- A **structure update plugin** which creates a basic data model for a water utility and modifies it in a series of four version updates. This plugin is located in the files *StructureUpdatePlugIn.cs*, *Version10000.cs*, *Version10001.cs*, *Version10002.cs*, *Version10003.cs*, and *Version10004.cs*.
- An acquisition **workflow** for the creation of new network pipes. This plugin is located in the *AcquisitionWorkflows.cs* file. The sample also adds three categories of workflows using the `AddOrUpdateWorkflow` API method during the version update in the *Version10003.cs* file.
- A **feature rules plugin** enforcing the module's business rules - in this case, filling in attributes whever a feature is created or modified. This plugin is located in the *FeatureRules.cs* file.

- A **document plugin** that creates a new menu item that links to the pipe creation workflow. This plugin is located in the *DocumentPlugIn.cs* file.

A more complete description of vertical application modules is contained in the chapter Creating Application Modules (page 63).

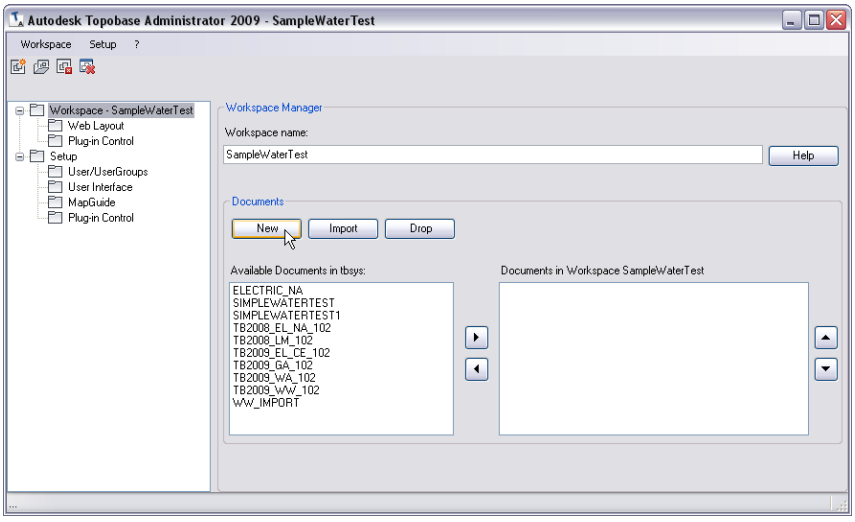
Procedure

Building and Installing

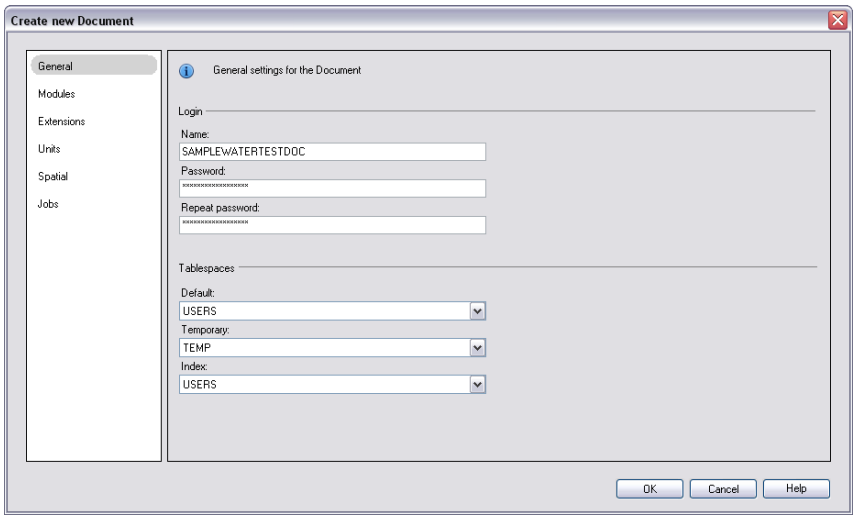
Build the sample project. The library (*VBSample117.dll* or *CSSample117.dll*) will automatically be placed in the Topobase Client *bin* directory. This library and the associated *.tbp* file need to be copied into the Topobase Administrator *bin* directory as well.

Creating a Document Using the Module

1. Launch the Topobase Administrator application.
2. Load an existing workspace or create a new workspace (see Create a Workspace (page 152) for more information).
3. Select the New button in the Document box to create a new document.

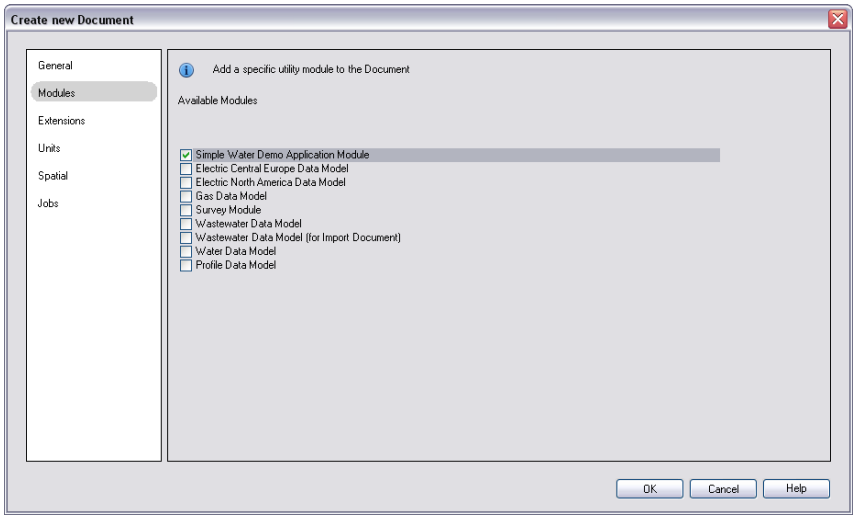


4. Enter a valid password in the Document tab of the Create New Document dialog box.



5 Select the Module tab on the left side of the Create New Document dialog box.

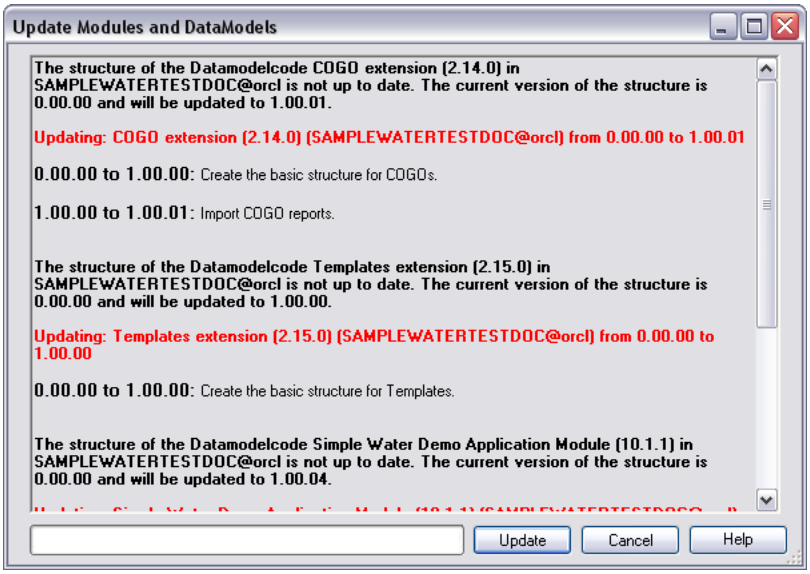
6 Check Simple Water Demo Application Module from the list of available modules.



NOTE If Simple Water Demo Application Module is not in the list, it means the Sample 117 *.dll* and *.tbp* files are not in the Topobase Administrator *\bin* directory. Close the Administrator, copy the files from the Topobase Client *\bin* directory, restart the Topobase Administrator, and try again.

7 Press OK. This will begin the process of creating a new document.

8 After a series of status message boxes, you will see the Update Modules and DataModels dialog box listing all the updates specified in the structure update plugin of Sample 117. Press Update.



9 This will create the data model and apply all the updates specified in the structure update plugin. When the process is complete, press Close.

10 The document based on the simple water module has now been created in the workspace. Select the Data Model and Workflows tree elements to see the data model and workflows provided by the module.

Sample 118 - Electric Templates

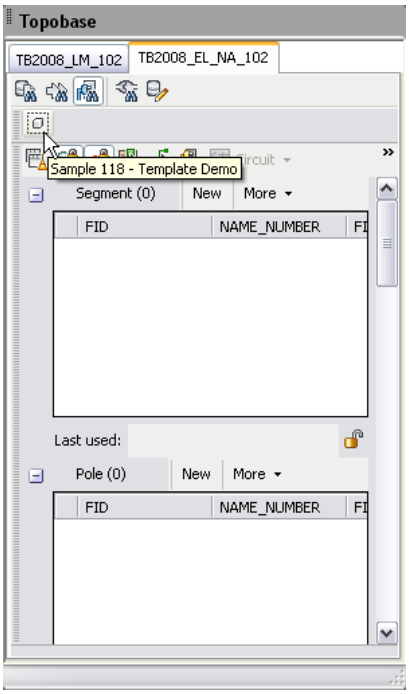
Purpose

This sample shows how to use API methods in the Electric Module to create and use electric templates. This sample requires a workspace that includes electric components.

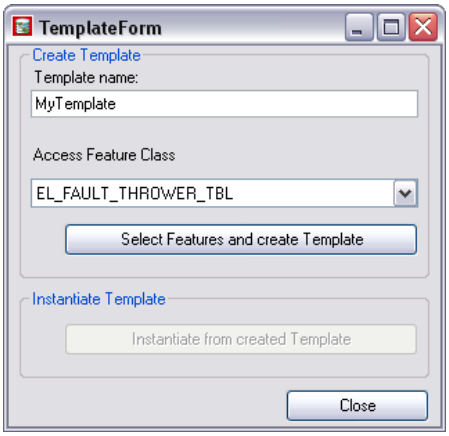
Procedure

To use this sample:

1. Build the sample. The plugin (*VBSample118.dll* or *CSSample118.dll*) will be placed in the Topobase Client *bin* directory. The *.tbp* file will also be copied to the Topobase Client *bin* directory.
2. Start Topobase and open the workspace.
3. Click Generate Graphic.
4. Click the document toolbar button with the caption Sample 118 - Template Demo. The button is only displayed when the tab for the document containing electric components has focus.



This will display the Sample 118 form:



5 Type in the name of the new template in the topmost text box.

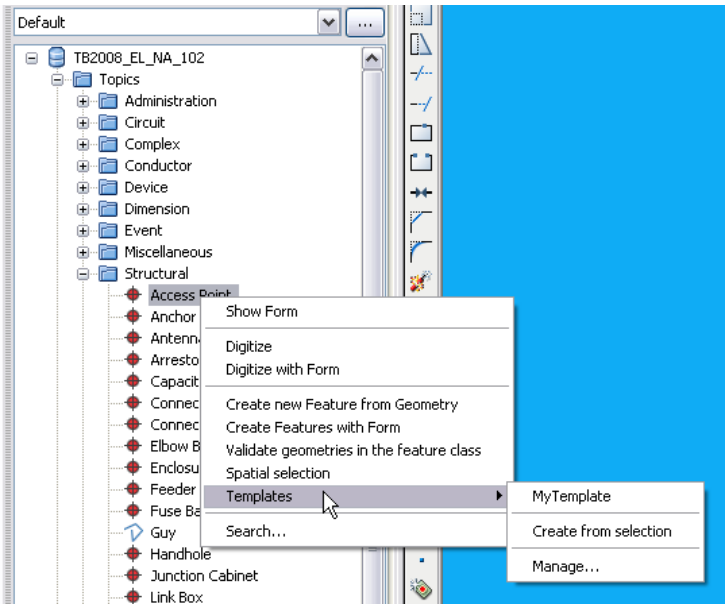
6 Select the feature class that the template will be attached to from the Access Feature Class combo box.

7 Press the Select Features And Create Template button. This will begin the process of creating a template.

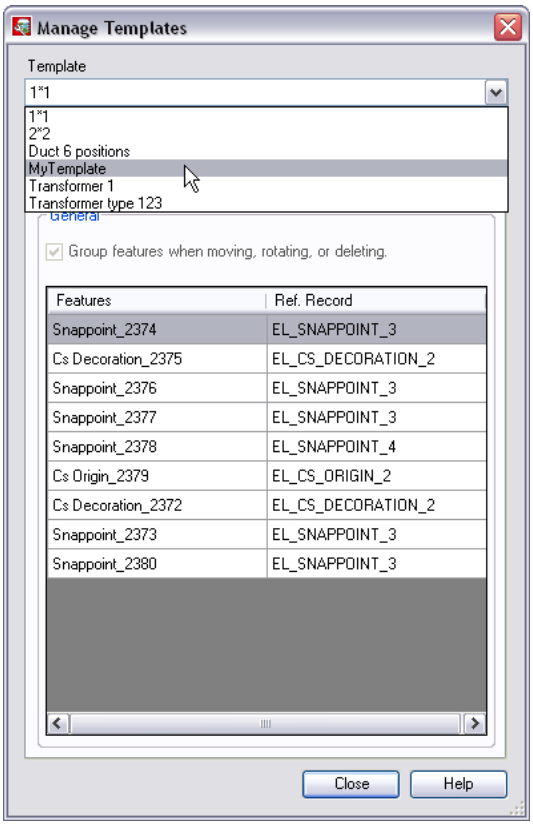
8 Select the electrical components to be added to the template. When you are done, press the Enter key.

9 Select the origin point and the orientation point for the template. This forms a reference for the placement and orientation of instances created from this template.

10 The template is now created and is assigned to the specified feature class. You can see this template by accessing the template explorer for the feature class. For example, if you created a template called "MyTemplate" assigned to the feature class "Access Point" (the default settings), bring up the context menu for the Access Point feature class and select Templates ➤ Manage... .



In the Manage Templates dialog box you will find the template MyTemplate among the list of templates.



11 To create new elements based on this template, press the Instantiate from created Template button on Sample 118's form.

12 Select the origin point and orientation point for the new instance. The locations of the origin and orientation points will define how the template components are positioned, scaled, and rotated for the new instance.

Sample 119 - Survey Import Plugin

Purpose

This sample shows how to create a FileFormatPlugIn component which is used to load custom formatted files in the Topobase Survey wizard. This sample requires a workspace with a survey document.

Procedure

To use this sample:

1. Build the sample. The plugin (*VBSample119.dll* or *CSSample119.dll*) will be placed in the Topobase Client *bin* directory. The *.tbp* file will also be copied to the Topobase Client *bin* directory.
2. Start Topobase and open the workspace.
3. Click Survey in the ribbon. This will bring up the Autodesk Topobase Survey wizard.
4. In the first page of the wizard, in the section Select a Calculation Project, create a new calculation project. First, select the Manage Calculation Projects button by the combo box. This will bring up the Manage Projects dialog box. Select the Create button and enter the new project's name. Select the new project in the Manage Projects list and select the Close button.
5. Select the Next button or "1. Calculation Project Settings" on the left of the dialog to go to the next page of the wizard.
6. Select the Next button or "2. Import Measurements" on the left of the dialog to go to the next page of the wizard.
7. In the Import Measurements page of the wizard, select "Sample Data Format" from the File Format combo box. This indicates we will use Sample 119 to load the data file.
8. Load one of the Sample 119 data files by selecting the Add button by the Files to import listbox. In the file dialog box, select either *Test1.demo* or *Test2.demo* from the *TestData* folder in the *119_SurveyImportPlugin* directory where the sample source code is located. *Test1.demo* contains simulated data from traditional survey techniques and *Test2.demo* contains simulated GPS coordinates.

9 In the Instrument combo box, select “Tachymeter 1” if you are loading *Test1.demo* or “GPS 1” if you are loading *Test2.demo*.

10 Select the Next button or “3. Find Reference Points” on the left of the dialog. The simulated data will be loaded into your document.

Sample 120 - Custom URL Invoking

Purpose

This sample shows how to use an ApplicationPlugin to create a new command that can be run using URL invoking.

Procedure

To use this sample:

1 Build the sample. The plugin (*VBSample120.dll* or *CSSample120.dll*) will be placed in the Topobase Client *bin* directory. The *.tbp* file will also be copied to the Topobase Client *bin* directory.

2 Start Topobase and open a workspace.

3 Click Generate Graphic.

4

Sample 121 - Web Service

Purpose

This sample contains both a web service that communicates with a local installation of Topobase and a web client that allows remote users to perform Topobase actions on the server.

Web services require the installation of IIS (Internet Information Services) on the machine. This can be done in the “Add/Remove Windows Components” section of the “Add or Remove Programs” Control Panel applet.

Procedure

To use this sample:

1. Load the solution for Sample 121 into Visual Studio 2008. You will see two projects, *CSSample121_Server* and *CSSample121_Client*.
2. Build the server project. This will create the files needed for the web service in the *\CSSample121_Server\Bin* directory.
3. Copy the following directories from *\CSSample121_Server\Bin* into the *C:\Inetpub\wwwroot* directory:
   - *App_Code*
   - *App_GlobalResources*
   - *App_WebReferences*
   - *bin*

   (Some of these folders may already exist. In those cases, copy the files into the folders of the same name.)

   Copy the *Service.asmx* file to the *C:\Inetpub\wwwroot* directory. If you have no other web services on this machine, copy the *Web.Config* file to the *C:\Inetpub\wwwroot* directory as well. If you have other web services on this machin, then only copy the item within the <appSettings> key within the *Web.Config* file to the existing Web.Config file in the *C:\Inetpub\wwwroot* directory.
4.

Sample 122 - Custom Point Numbering

Purpose

This sample demonstrates both adding a page in the Topobase Administrator using the new PointNumberGroupTreeView control and how to add feature rules in the Topobase Client to insert custom point numbers. This sample requires a workspace that includes feature classes with a property named TB_POINTNUMBER.

Procedure

To use this sample:

1. Build the sample. The plugin (*VBSample122.dll* or *CSSample122.dll*) will be placed in the Topobase Client *bin* directory. The *.tbp* file will also be copied to the Topobase Client *bin* directory.
2. Copy all of these files (either *VBSample122.dll* and *VBSample122.tbp*, or *CSSample122.dll* and *CSSample122.tbp*) to the Topobase Administrator *bin* directory as well. This sample contains both a DocumentAdminPage plugin for the Administrator and a Feature Rule plugin for the Client.
3. Start Topobase Administrator and load a workspace which has feature classes containing a property named TB_POINTNUMBER.
4. In the workspace tree view under the nodes for each document you will see a new node named "My Point Numbering". Select this to bring up the My Point Numbering page.
5. In the My Point Numbering page select the New button and create a new numbering group. Select one or more feature classes that contains a TB_POINTNUMBER property (such classes will be in black, not red) and drag it from the "Unassigned FC" section of the tree to the new group. Exit the Topobase Administrator.
6. Start the Topobase Client and open the same workspace.
7. Select Generate Graphic and generate the graphics for the workspace.
8. Make sure that the selected document in the task pane is the same as the document you modified using the My Point Numbering page in Topobase Administrator.
9. Using the Document Explorer, expand the topics until you find a feature class you added to the new numbering group above.
10. Digitize new points for this feature class.
11. Select the new points and view their attributes. In the table, you will see the new points have been assigned new point numbers starting with the value "MySample0000001" and automatically incrementing for each new point.

Sample 123 - Wizard Using WPF

Purpose

This sample shows how to use the Windows Presentation Foundation (WPF) to design a Topobase Wizard.

Procedure

To use this sample:

1. Build the sample. The plugin (*VBSample123.dll* or *CSSample123.dll*) will be placed in the Topobase Client *bin* directory. The *.tbp* file will also be copied to the Topobase Client *bin* directory.
2. Start Topobase and open a workspace.
3. Click Generate Graphic.
4. In the task pane, right click on the root node in the Document Explorer to bring up the context menu. In the context menu, select Samples ➤ Sample123 Wizard.

   This will bring up the Sample123 Wizard form.
5. In the first page of the wizard, select either line measurement or area measurement. Select Next.
6. Type in the directory and filename where you wish to save the text report, or select the browse button and choose the directory and filename using the Open File dialog box. You can chose an existing text file, or use the name of a file that does not yet exist. Select Next.
7. The final page of the wizard will show the result of the computation in a text box. The information will also be appended to the text file chosen in the second page of the wizard. Select Close to exit the wizard.

TBP File Format

A

Introduction

Autodesk® Topobase is based on a plugin concept that allows developers to extend Topobase for their own needs.

The plugin model serves as the basis for adding features to the user interface, creating workflows, setting business rules, updating the data model, and more. Many of the standard Topobase components are built using plugins.

If you want to create your own plugin, you have to create a *.dll* with one or more plugin classes in it and a *.tbp* file which tells Topobase which plugin classes are implemented and where to find them. For more information about creating a plugin, see Quick Guide: Creating a Topobase Plugin (page 51).

You can also create one.*tbp* file for two or more *.dll*'s.

Example TBP Files

This example shows the contents of a *.tbp* file exposing three different plugin classes from the library *VbSample.dll*:

```
<?xml version="1.0" encoding="utf-8"?>
<PlugIn>

<ApplicationPlugIn ClassName="MyApplication1"
   AssemblyName="VbSample.dll" Namespace="VbSample"
   DocumentKey="" MapName=""  Priority="100"
   ExecutionTargetWeb="True" ExecutionTargetDesktop="True"
   Company" Author="Hans Maiser"/>
<ApplicationPlugIn ClassName="MyApplication1" Company="My
   DocumentKey="" MapName=""  Priority="100"
   AssemblyName="VbSample.dll" Namespace="VbSample"
   ExecutionTargetWeb="True" ExecutionTargetDesktop="True"
   Sample Company" Author="Hans Maiser"/>
<DocumentPlugIn ClassName="MyDocument"
   AssemblyName="VbSample.dll" Namespace="VbSample"
   DocumentKey="" MapName=""  Priority="100"
   ExecutionTargetWeb="True" ExecutionTargetDesktop="True"
   Company="My Sample Company" Author="Hans Maiser"/>

</PlugIn>
```

```
<?xml version="1.0" encoding="utf-8"?>
<PlugIn>

<ApplicationPlugIn ClassName="MyApplication1"/>
<ApplicationPlugIn ClassName="MyApplication2"/>
<DocumentPlugIn ClassName="MyDocument"/>

</PlugIn>
```

You can use the `Default` tag to define properties used by all plugin classes in the library. To overwrite a sinlge property, include the correct attribute within the plugin tag.

The following example is the same *.tbp* file rewritten to use the `Default` tag:

```
<?xml version="1.0" encoding="utf-8"?>
<PlugIn>

<Default
   AssemblyName="VbSample.dll"
   Namespace="VbSample"
   DocumentKey=""
   MapName=""
   Priority="100"
   ExecutionTargetWeb="True"
   ExecutionTargetDesktop="True"
   Company="My Sample Company"
   Author="Hans Maiser"
/>

<ApplicationPlugIn ClassName="MyApplication1"/>
<ApplicationPlugIn ClassName="MyApplication2" Author="Heini
Dumpf"/>
<DocumentPlugIn ClassName="MyDocument"/>

</PlugIn>
```

Plugin Tag Details

The following table lists all plugin tags:

<ApplicationPlugIn>	Plugin that adds a menu item or toolbar item to the workspace area of the task pane. The plugin class is derived from the `Topobase.Forms.ApplicationPlugIn` abstract class.
<DocumentPlugIn>	Plugin that adds a context menu item or toolbar item to a document tab in the task pane. The plugin class is derived from the `Topobase.Forms.DocumentPlugIn` abstract class
<DialogPlugIn>	Plugin that adds user interface components to forms. The plugin class is derived from the `Topobase.Forms.DialogPlugIn` abstract class
<ApplicationOptionPage>	Plugin that creates an option page for the application options. The plugin class is derived from the `Topobase.Forms.OptionPages.ApplicationOptionPage` abstract class

<DocumentOptionPage>	Plugin that creates an option page for the document options. The plugin class is derived from the `Topobase.Forms.OptionPages.DocumentOptionPage` abstract class
<ApplicationFlyIn>	Plugin that adds a moveable, dockable window to the task pane. The plugin class is derived from the `Topobase.Forms.Desktop.ApplicationFlyIn` or `Topobase.Forms.FlyIns.ApplicationFlyIn` abstract class
<DocumentFlyIn>	Plugin that adds a moveable, dockable window to a document tab within the task pane The plugin class is derived from the `Topobase.Forms.Desktop.DocumentFlyIn` or `Topobase.Forms.FlyIns.DocumentFlyIn` abstract class
<DialogFlyIn>	Plugin that adds a docked, unmoveable window to forms. The plugin class is derived from the `Topobase.Forms.Desktop.DialogFlyIn` or `Topobase.Forms.FlyIns.DialogFlyIn` abstract class
<WorkflowPlugIn>	Plugin for a workflow which may include user-interface elements shown in the task pane. The plugin class is derived from the `Topobase.Workflows.WorkflowPlugIn` abstract class
<FeatureRulePlugIn	Plugin that defines business rules. The plugin class is derived from the `Topobase.FeatureRules.FeatureRulePlugIn` abstract class
<DocumentStructureUpdatePlugIn>	Plugin that modifies the data model. The plugin class is derived from the `Topobase.Update.DocumentStructureUpdatePlugin` abstract class.
<ReportPlugIn>	Plugin that adds a report generator. The plugin class is derived from the `Topobase.Forms.ReportPlugIn` abstract class.

Property Details

The following table details all possible properties.

AssemblyName	Name of the *.dll* file, not including the path.

Namespace	The Namespace name of your *.dll* (as defined in the C# or VB project). Usually the same as the AssemblyName property.
ClassName	Name of the plugin class.
MapName	Name of the map interface. The default is "" (empty quotation), which means the plugin runs with the default Autodesk connection (that is, Autodesk Map 3D 2008 or Autodesk MapGuide Enterprise 2008.) If you want to use your own application for the map interface, specify the name for this property.
Priority	Defines the order in which the plugins are loaded. For example, a plugin class with a priority of 50 is loaded before a class with a priority of 100. The recommended value is 100. If you require a range, use values between 50 and 150.
ExecutionTargetWeb	If your plugin is designed to work in Topobase Web, set this property to “true”, otherwise set this property to “false”. Plugins that use standard Windows Forms and controls cannot be used in Topobase Web.
ExecutionTargetDesktop	If your plugin is designed to work in Topobase Client, set this property to “true”, otherwise set this property to “false”.
Company	Name of the company that built the plugin.
Author	Name of the plugin developer. The Company and Author properties are shown in the list of plugins and also displayed if an error occurs in the application.
DocumentKey	(Obsolete - use the DMCode property instead.) Used to specify the documents that will cause the plugin to load. Default is "" (empty quotation), which indicates that your plugin will be loaded with every Topobase document. If you want the plugin to only load with a specific Topobase document, set this property and add the same key to the TB_GN_DOCUMENT_KEY table.
ApplicationKey	(Obsolete - use the DMCode property instead.) Similar to the DocumentKey but is an application-wide setting. Add the application key to the TB_GN_APPLICATION_KEY table in TBSYS User.

DeactivationKey	Enables you to “switch off” a plugin or group of plugins. Use the "Options-License" property. Use the same DeactivationKey name for all plugins that belong to the same functional group.
DMCode	Used to specify which modules the plugin will load in. By default the plugin will load with documents based on any kind of module. DMCodes are stored in the TB_DATAMODELCODE table. The DMCode format is “X.X.X”, where the first number represents the company, the second is the module (for example, Wastewater), and the third is the submodule (for example, Wastewater Switzerland). Modules not developed by Autodesk will have a value between 3 and 99 for the company code.

Other Properties

To specify that a dialog plugin is only active with a specific form, set the `Name` tag in the.*tbp* file. In the following example, the dialog plugin MyPointClass will only be visible in the form/table “MyPointName”:

```
<DialogPlugIn ClassName="MyPointClass" Name="MyPointName" />
```

NOTE If you set `Name=""` then the dialog plugin is started with all forms.

TBP Examples

For more information about TBP and sample code example, see Developer Samples (page 147).

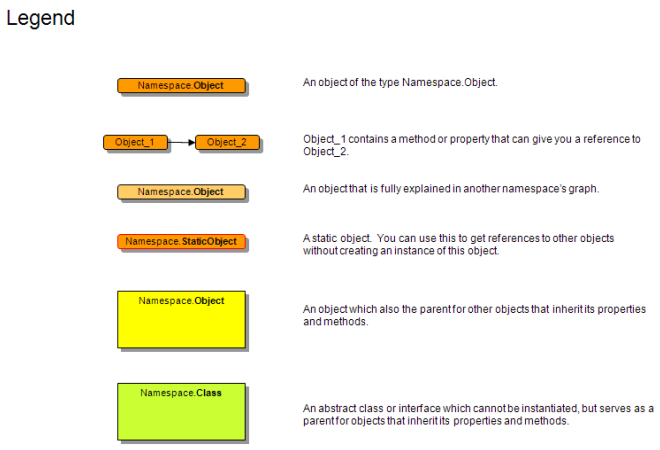
Object Models

B

Legend

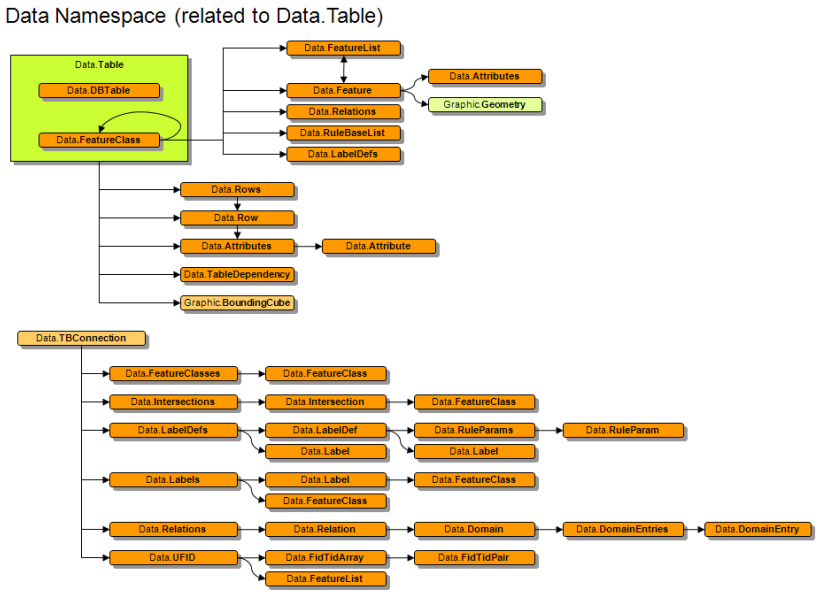
The following object model images represent the hierarchical links of the major objects and abstract classes or interfaces for the Framework and Workflows APIs.

For more information about the objects, see the Topobase API Reference documentation.

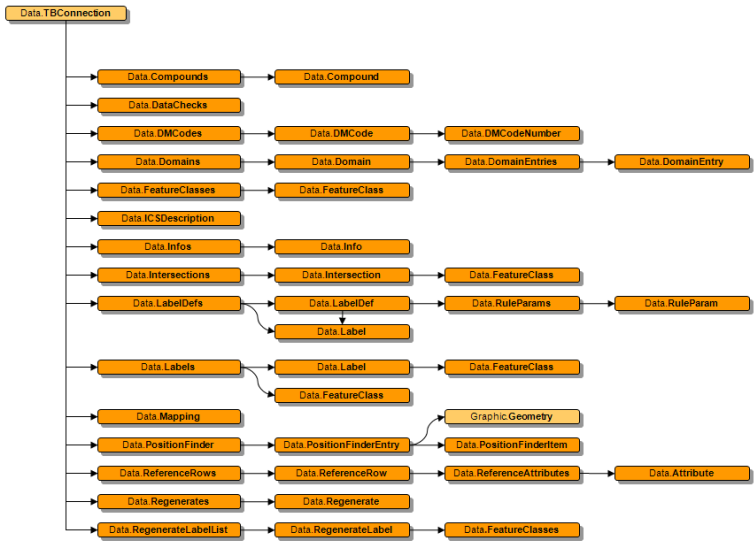


Framework

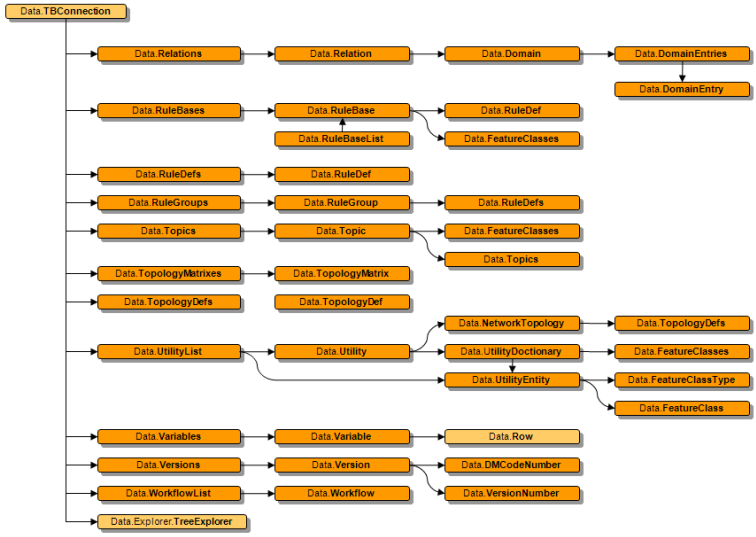
Data Namespace



Data Namespace (other connections from TBConnection 1)

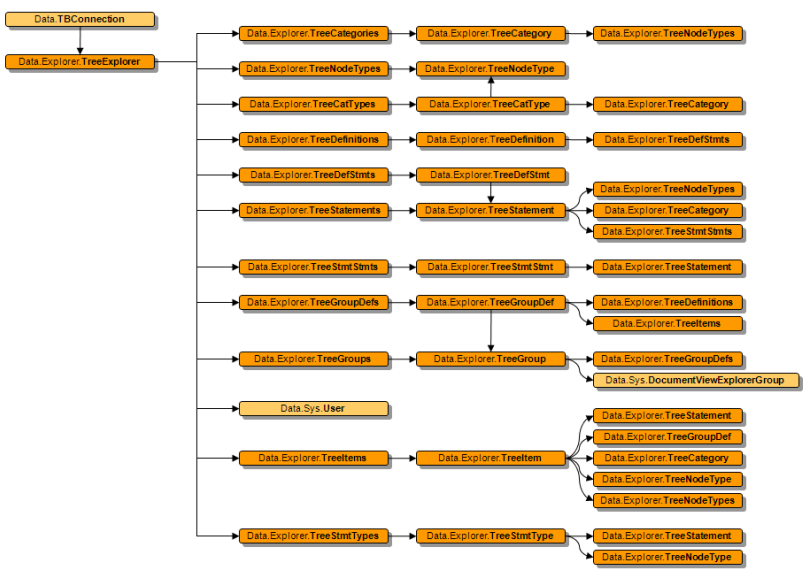


Data Namespace (other connections from TBConnection 2)

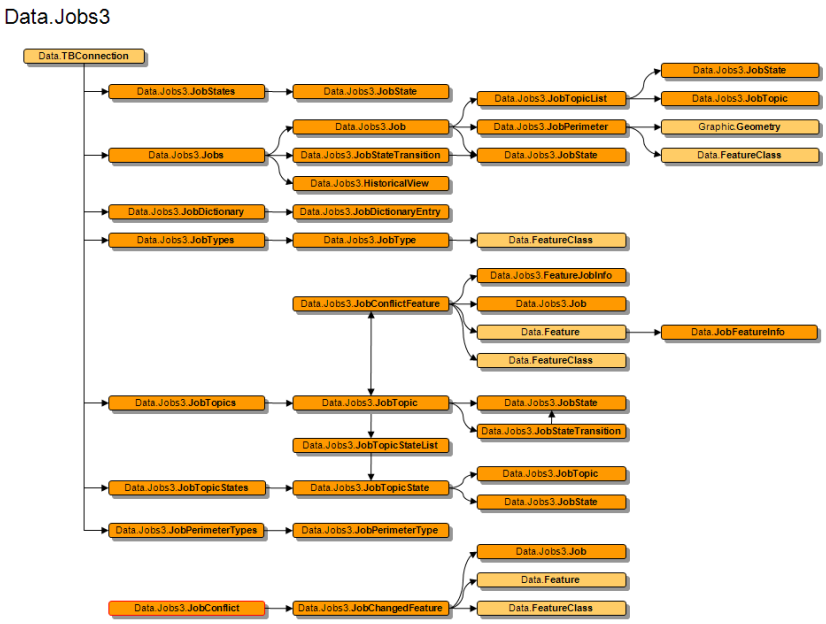


Data.Explorer Namespace

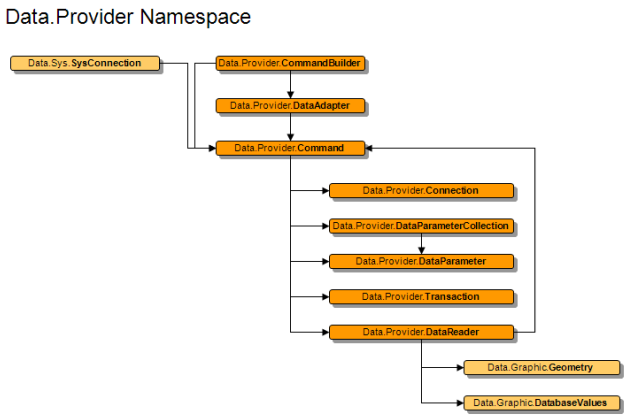
Data.Explorer Namespace



Data.Jobs3 Namespace

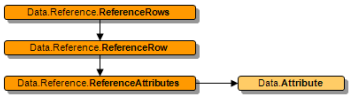


Data.Provider Namespace



Data.Reference and Data.Util.OperationLog Namespaces

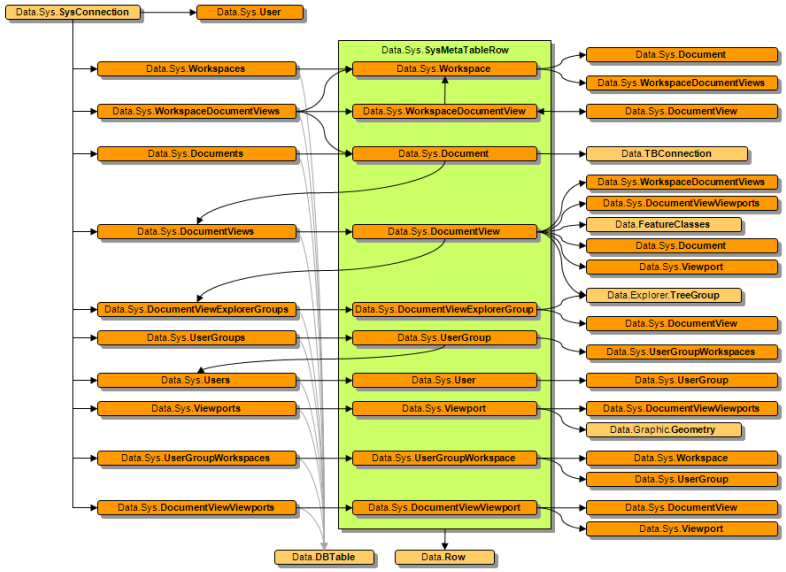
Data.Reference Namespace



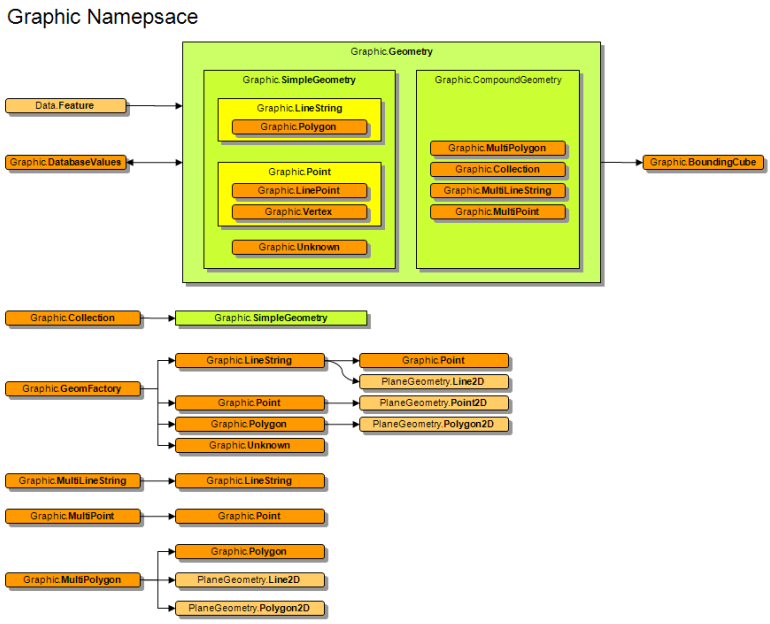
Data.Util.OperationLog Namespace

Data.Sys Namespace

Data.Sys Namespace

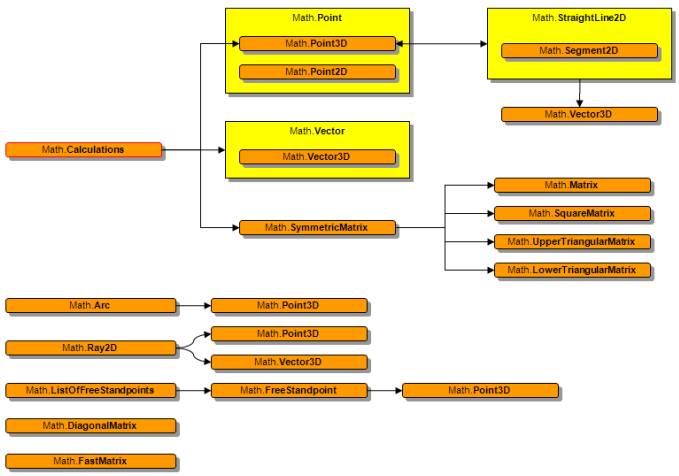


Graphic Namespace



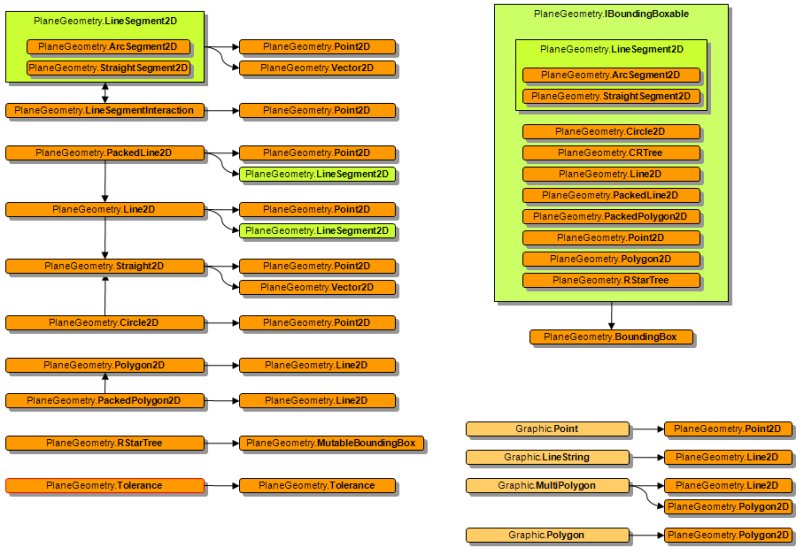
Math Namespace

Math Namespace



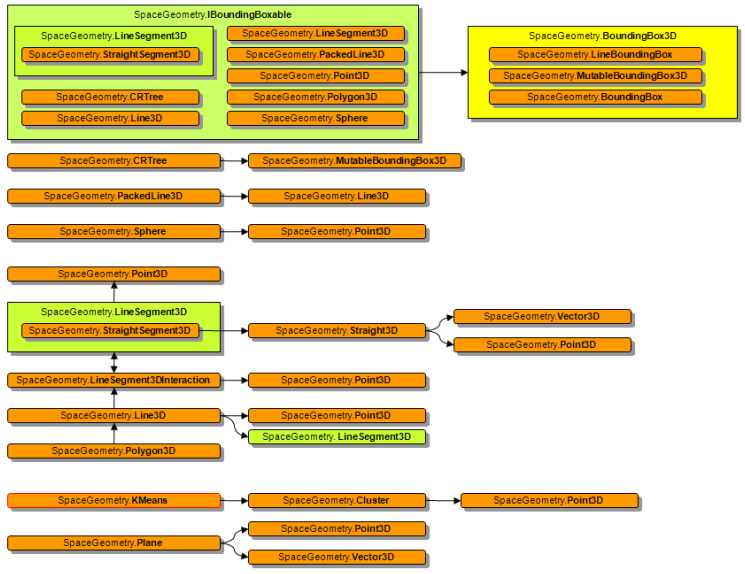
PlaneGeometry Namespace

PlaneGeometry Namespace

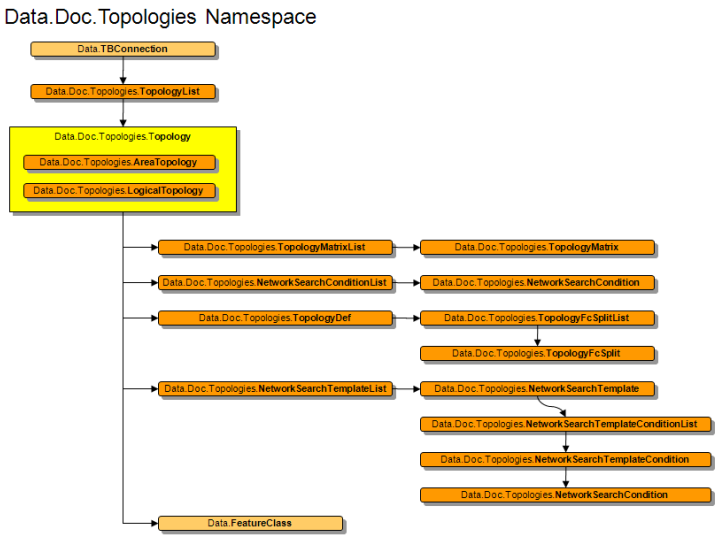


SpaceGeometry Namespace

SpaceGeometry Namespace

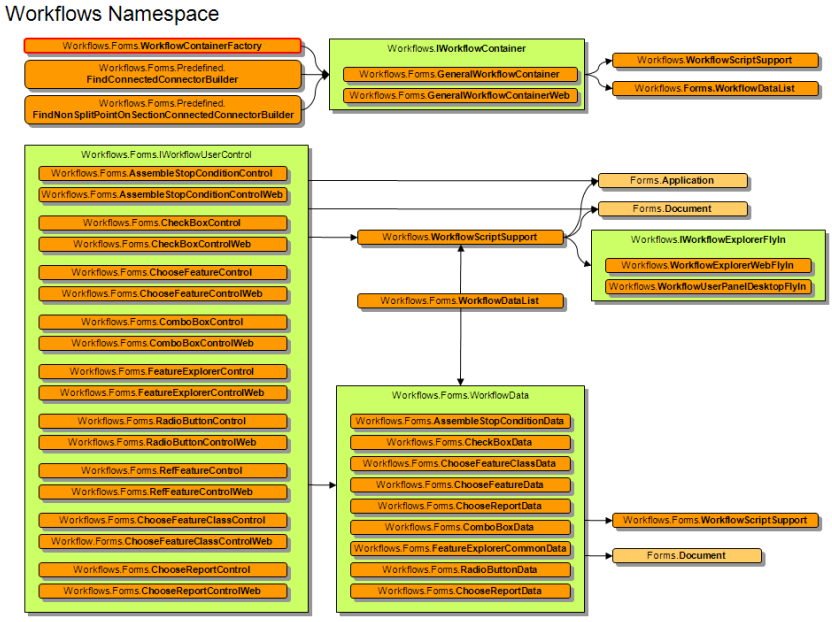


Data.Doc.Topologies Namespace



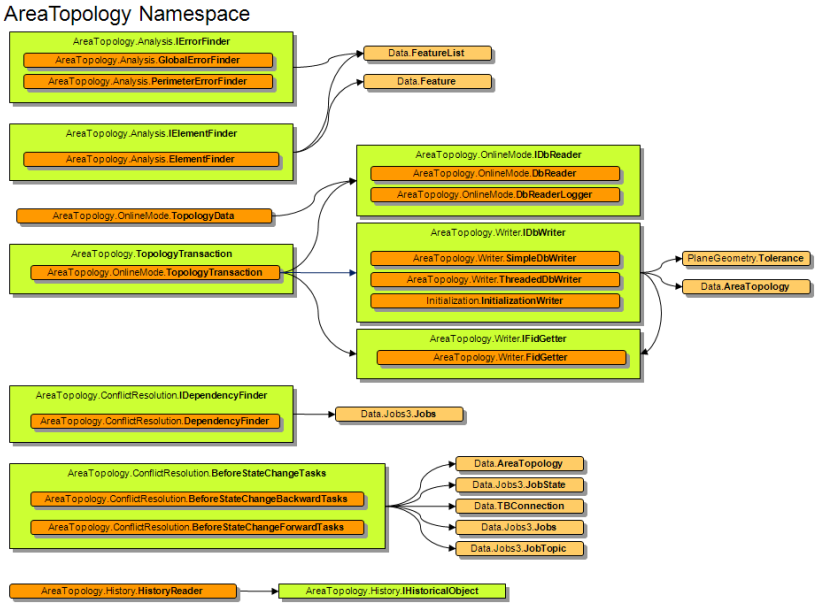
Workflows

Workflows Namespace

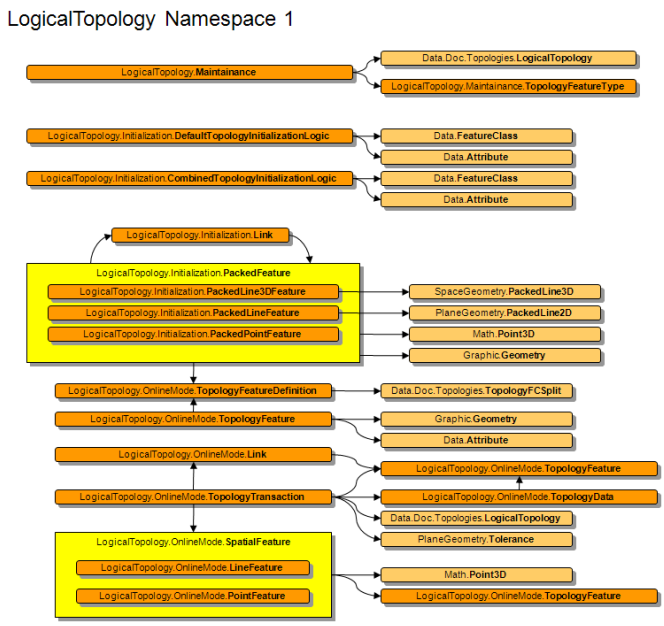


Area Topology

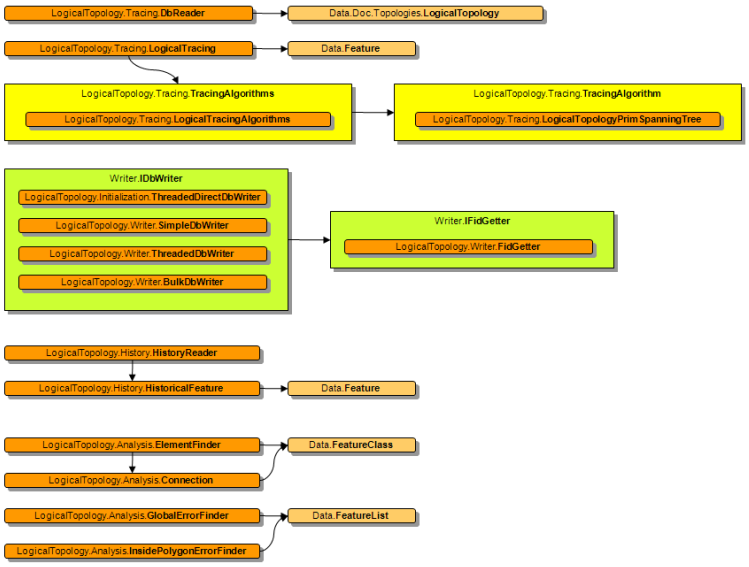
AreaTopology Namespace



LogicalTopology Namespace



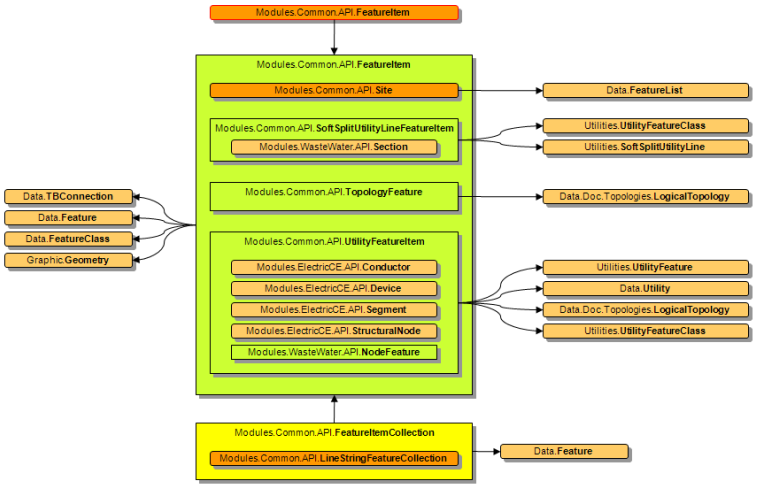
LogicalTopology Namespace 2



Vertical Application Modules

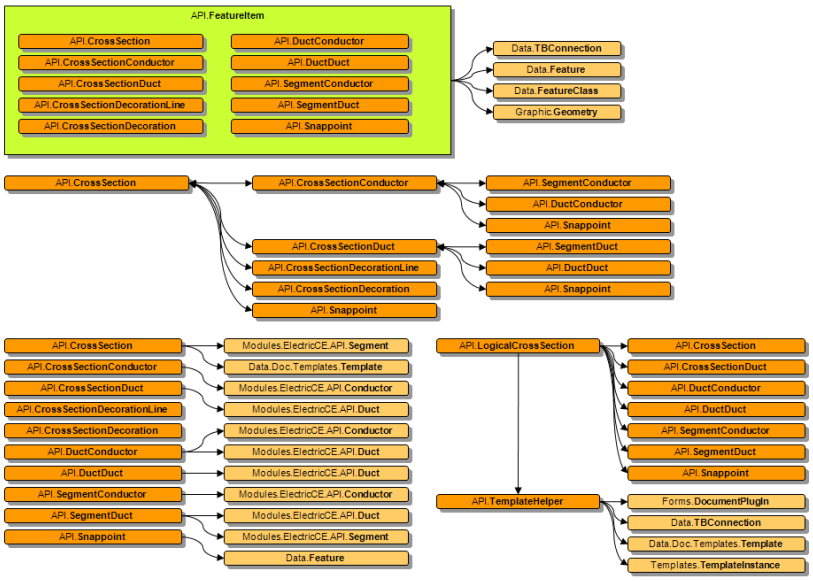
Common Namespace

Modules.Common Namespace



Electric.Common Namespace

Modules.Electric.Common Namespace



Index

A

API Reference 1
application flyin sample 229
application options sample 199
application toolbar sample 231
assembly information 57
Autodesk Topobase Administrator Guide 1

B

batch files 260

C

canvas control sample 237
checked list box sample 230
class call via batch file sample 260
client side feature rules sample 268
code samples
  introduction 147
Conditional loading of plugin 290
confirm box sample 176
connection tools sample 207
control text sample 218
controls sample 178
create a drawing template 155
create a workspace 152
create all dialog boxes sample 192
create structure sample 149
create topobase user sample 263

D

date/time picker sample 202
desktop sample 219
dialog box control update sample 238
dialog box flyin sample 223
dialog box user controls sample 214
dialog box validation sample 194
dialog button sample 171
dialog events sample 175
dialog menu sample 174
dialog toolbar button sample 166
directory text box sample 203
dock in container sample 204
DockableForm 204
DockInContainer 204
document context menu sample 163
document flyin sample 179
document options sample 213
document toolbar button sample 164
drawing template, create a 155
drop down buttons sample 210
DWT (drawing template) 155

F

feature explorer sample 266
feature information sample 187
features sample 254
file download sample 188
file text box sample 203
file upload sample 188
flyin container sample 226
flyin sample 179, 223, 229
forms sample 178

G

generic dialog box sample 214
grid control sample 193

H

Hello World 2, 52
highlight sample 186

I

information sample 187
input box sample 176
interaction sample 197
introduction 1

J

jobs sample 249

L

list box multi selection sample 232
list box sample 185, 211, 230
list box with context menu sample 181

M

main menu sample 167
main toolbar sample 169
map button sample 183
map function sample 257
matrix control sample 196
menu control sample 234
menus sample 196
Microsoft Visual Studio 2005 51–52
multiple input box sample 176

O

ODP .Net sample 161
options page with tree nodes sample 224
options sample 213
oracle data provider .Net sample 161
overview 1

P

passwords 148
picture combobox sample 259
plugin
    details 288
    example 285
    file format 285
    test 60
plugin definition file 59
plugins 2
    introduction 51, 285
progress bar sample 181
project
    add references 56
    default class 57
    run from Visual Studio 55
    write to bin folder 54
project, create a 52

R

read/write features sample 159
remote control sample 240
reports sample 221

S

samples
    building 148
    common steps 149
    details 149
    introduction 147
    running 149
    usernames and passwords 148
samples, developer 147
samples, preparing to run the 2
script engine sample 262
settings sample 213
special menu items sample 212
SQL export sample 233

T

tab control sample 236
TBBatch.exe 260
TBP 2, 59, 147, 285
    details 288
    example 285
    file format 285
TBSAMPLE workspace 152
TBSYS 148
toolbar sample 231

toolbars sample 196
Topobase
    marketing description 2
Topobase Database Server 148
Topobase user sample 263
tree nodes sample 224
tree view sample 182
treeview with context menu sample 201

U

unit support sample 264
update plugin sample 244
usernames 148

W

web browser sample 235
Windows menu items sample 189
workflows sample 242
workspace, create a 152
workspace, demo 52