Setting up Visual studio 6 IDE for MDL programming

by Stanislav Sumbera

Introduction

MicroStation Development Environment (MDE) presented on installation of MicroStation unfortunately doesn't contain any Integrated Development Environment for developing MDL programs. Although you may write MDL code in any text editor, I guess MDL programmers in the 21st century deserve more, at least :

- File management
- Syntax highlighting
- Automatically Completing Statements - Intellisense
- Context help
- Comprehensive visual debugger
- Optionally version management for large projects

Using these features will certainly help you in developing process and will positively affect the quality of your MDL application. Although you may use any IDE I would recommend to use Visual Studio v.6 or Visual Studio.NET respectively. The main reason is that you will get with Visual Studio compiler used by MicroStation make files to compile MDL into native code (DLL).

Setting up Visual Studio v.6

Make sure you have installed Visual C++ v.6 with service pack 5 and Microsoft Development Library (MSDN). To set up IDE for MDL several steps need to be done. We will go through them. step by step.

Step 1. MDL language syntax highlighting

MDL code is C code. We need Visual Studio to recognize file extensions of MDL related files as C-based files.

These extensions include `.mc`; `.fdf`; `.r`; `.mt`;. The file extension information is stored in the registry key: See code below.

Run file **VC_MDL_HIGHLIT.reg** from download package for this article to update registry entries or extend registry manually by running regedit.exe.

- Now run Visual Studio, open any `.mc` file via **File/ Open** menu and see highlighting syntax of code.
- Go to **Tools/Options**... menu and select **Format** tab page.
- In the **Category** list select `Source Windows`; in the **Font** option list choose `Courier New` font with size `10pt`.
- Select in **Colors** option list type of code text and use **Foreground** and **Background** items to set its corresponding colors as you wish. Check the `Keyword` type to be colored.

Step 2. Set MDL built- in functions to be highlighted as keywords

Wouldn't be nice to have highlighted all MDL built-in functions as keyword? We may do this by populating file **usertype.dat** with all keyword names - in this case all mdl functions. All MDL API functions (even undocumented) may be listed by runing **mcomp-zb**.

The file **usertype.dat** must be subsequently copied into the same directory where msdev.exe resides. By default, this will be *C:\Program Files\Microsoft Visual Studio\Common\MSDev98\Bin.* You may use the file in download package. Optionally, you may add there the MDL types or constants. Then restart Visual Studio, open an arbitrary `.mc` file and see keyword-color of all mdl functions.

```
[HKEY_CURRENT_USER\Software\Microsoft\DevStudio\6.0\
Text Editor\Tabs/Language Settings\C/C++]
"FileExtensions"="cpp;cxx;c;h;hxx;hpp;inl;tlh;tli;rc;rc2;mc;fdf;mt;r"
```

Step 3. Creating MDL project

- Select **File/New** to bring up the list shown in Figure 1 below.

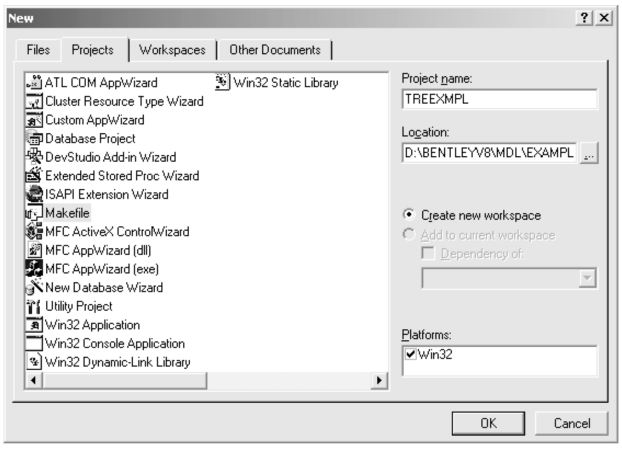

New
Files | Projects | Workspaces | Other Documents
ATL COM AppWizard
Cluster Resource Type Wizard
Custom AppWizard
Database Project
DevStudio Add-in Wizard
Extended Stored Proc Wizard
ISAPI Extension Wizard
Makefile
MFC ActiveX ControlWizard
MFC AppWizard (dll)
MFC AppWizard (exe)
New Database Wizard
Utility Project
Win32 Application
Win32 Console Application
Win32 Dynamic-Link Library
Win32 Static Library
Project name: TREEXMPL
Location: D:\BENTLEY\V8\MDL\EXAMPL
Create new workspace
Add to current workspace
Dependency of:
Platforms: Win32
OK Cancel

***Figure 1** The New dialog*

- Select **Project** tab page. From the available projects, choose `Makefile`. On the right side of the dialog, in the **Location** edit item, select the parent folder of you project
- Type your project name into the **Project name** edit item. Visual Studio will make the folder according to the Project name (or will use the existing one if the folder with this name already exists).
- Toggle On option **Create new workspace**. Press ***OK***.

Next wizard dialog asks for **Command line** for DEBUG configuration. Visual Studio has basically two configuration for running compilation process: Debug used for debug compilation, and Release, used for final release code.

You may use them for compilation, however I would recommend to use only one configuration and directly edit batch and make file for any necessary changes in build process. Delete any content there and type in your project name plus .bat extension, for instance treexmpl.bat. This will be a batch file for running MicroStation compilation.

Then type into the Output your final application name, e.g. treexmpl.ma, and you may leave **Rebuild All Switch** as it is. Press `Finish` button.

Wizard creates workspace and project with three virtual folders: Source Files, Header Files and Resource Files. Now you may add source files to appropriate folders by clicking right mouse on item and selecting **Add Files to Folder**.

After adding all files you should be able to see Class View with all structures and functions used in your project.

This is very useful for navigation through the code. If Wizard Bar tool box is presented on upper pane (see menu **Tools/Customize/Toolbars**) you may then quickly access function from option list there.

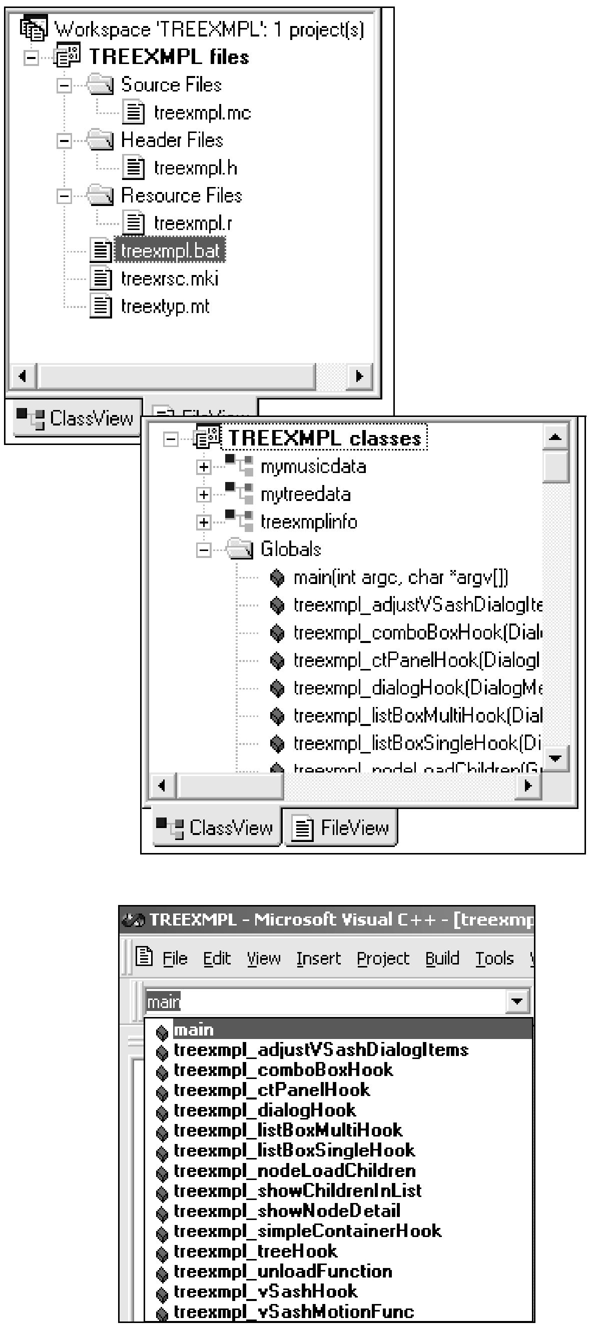
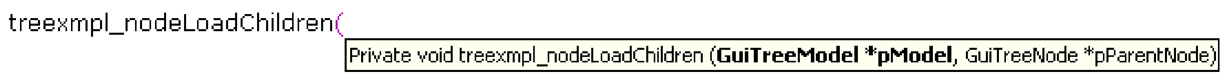

Workspace 'TREEXMPL': 1 project(s)
TREEXMPL files
Source Files
treexmpl.mc
Header Files
treexmpl.h
Resource Files
treexmpl.r
treexmpl.bat
treexrsc.mki
treextyp.mt
ClassView
TREEXMPL classes
mymusicdata
mytreedata
treexmplinfo
Globals
main(int argc, char *argv[])
treexmpl_adjustVSashDialogIte
treexmpl_comboBoxHook(Dial
treexmpl_ctPanelHook(Dialogl
treexmpl_dialogHook(DialogMe
treexmpl_listBoxMultiHook(Dial
treexmpl_listBoxSingleHook(Di
ClassView FileView
TREEXMPL - Microsoft Visual C++ - [treexmp
File Edit View Insert Project Build Tools
main
main
treexmpl_adjustVSashDialogItems
treexmpl_comboBoxHook
treexmpl_ctPanelHook
treexmpl_dialogHook
treexmpl_listBoxMultiHook
treexmpl_listBoxSingleHook
treexmpl_nodeLoadChildren
treexmpl_showChildrenInList
treexmpl_showNodeDetail
treexmpl_simpleContainerHook
treexmpl_treeHook
treexmpl_unloadFunction
treexmpl_vSashHook
treexmpl_vSashMotionFunc

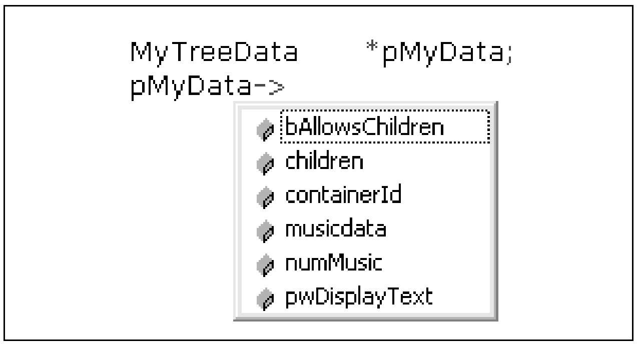
Figure 2

Visual Studio also helps with statement completition features (intellisense) of declared functions and types. If you have checked in **Tools/Options** dialog in **Editor** tab options for **statement completition** you may see parameter list when typing a function, for instance. **See Code 2 below**.

Code 2


```
treexmpl_nodeLoadChildren(
                          Private void treexmpl_nodeLoadChildren (GuiTreeModel *pModel, GuiTreeNode *pParentNode)
```

Or trying to access members of data structure:



```
MyTreeData    *pMyData;
pMyData->
```

Figure 3 *Data structure*

Step 4. Enabling intellisense for MDL built-in functions

Unfortunately intellisense does't work implicitly for built-in mdl functions. Wouldn't be this a nice feature when programming with MDL? Fortunately there is a trick how to force Visual Studio to show parameter info and data members of mdl functions. For each workspace Visual Studio stores browse information in binary `.ncb` file. Browse information hold records for each entity and where it's used in our project files.

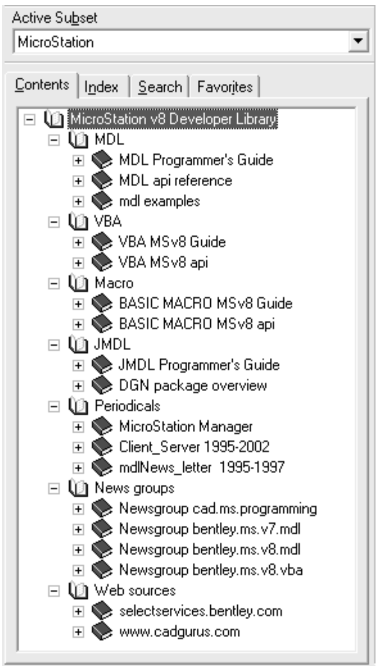
This data are used by intellisense to offer parameter info when typing a function. Intellisense works fine for function definition. MDL built-in function are declared in fdf files. So we need to temporarily change declaration to definition and include them into Visual Studio project so intellisense would catch them. I have prepared application **MDLtool** which will do all necessary steps for you to enable intellisense for your MDL project as may be seen in **Code 3 below**.

Step 5. Integrating MDL help with MSDN context help

Very often we need to search particular keyword or function definition in the Help documentation or in any other supplied help files. Visual Studio provides MSDN help containing all necessary information for developers under Windows platform. Since MDL is C based language we may find there, for example, a useful C code reference.

But I guess you would like to have there a MDL help reference and MDL programmer's guide help files too. Then we need to integrate supplied Bentley help files with MSDN help files. You may use for this purpose free help integration utility called **MSDNIntegrator** which is stored in download package to this article. Run this tool and specify **chm** and **chi** MDL help files. Only problem you need to overcome is how to obtain CHI file. CHI file may be re-generated from recompilation of original CHM file via HTML HELP workshop or other tool (like FAR).

We hope Bentley will soon deliver CHI files together with their CHM help files. In this manner you may build up your knowledge base containing all necessary information for software development. After integration run Visual Studio, enter any mdl function and while standing on the function with cursor pres F1. MSDN help will be displayed, after re-indexing its content a description for the function should be displayed. You may add other resources for rich knowledge base as is shown on figure below.



Active Subset: MicroStation

```
MicroStation v8 Developer Library
  MDL
    MDL Programmer's Guide
    MDL api reference
    mdl examples
  VBA
    VBA MSv8 Guide
    VBA MSv8 api
  Macro
    BASIC MACRO MSv8 Guide
    BASIC MACRO MSv8 api
  JMDL
    JMDL Programmer's Guide
    DGN package overview
  Periodicals
    MicroStation Manager
    Client_Server 1995-2002
    mdlNews_letter 1995-1997
  News groups
    Newsgroup cad.ms.programming
    Newsgroup bentley.ms.v7.mdl
    Newsgroup bentley.ms.v8.mdl
    Newsgroup bentley.ms.v8.vba
  Web sources
    selectservices.bentley.com
    www.cadgurus.com
```

Figure 4 *Help files*

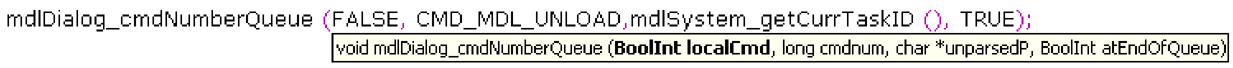
Step 6. MDL tool - advanced customization of Visual Studio

Macros are pretty cool in Visual Studio. Actually you may write your own VBScript macros to automate tasks when writing MDL code. Macros can reduce boring iocopy-paste-renamel. workflow in development process. Moreover you may create for your development team code templates and wizards to automate common steps.

As an example I have made simple prototype of such a functionality which you may freely use from the download package.

- Copy the **MDLtool.dll** into the directory *Common\MSDev98\AddIns* and copy templates to the *Common\MSDev98\template* directory respectively founded under Visual Studio base directory (by default it will be *C:\Program Files\Microsoft Visual Studio*).

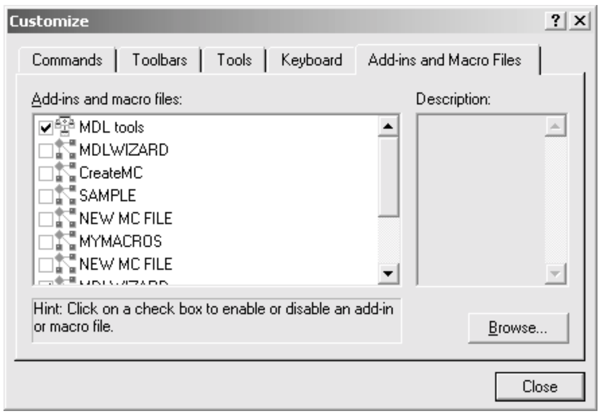
Code 3



```
mdlDialog_cmdNumberQueue (FALSE, CMD_MDL_UNLOAD,mdlSystem_getCurrTaskID (), TRUE);
                         void mdlDialog_cmdNumberQueue (BoolInt localCmd, long cmdnum, char *unparsedP, BoolInt atEndOfQueue)
```

- Restart you Visual Studio and go to **Tools/Customize** menu. The `Customize` dialog box will appear. Go to tab page **Add-ins and Macro files** and check in MDL tools in list item **Add-ins and macro files**.

An MDL toolbar will be displayed. Now you may create MDL application on 2 clicks!



Add-ins and macro files:
- [✓] MDL tools
- [ ] MDLWIZARD
- [ ] CreateMC
- [ ] SAMPLE
- [ ] NEW MC FILE
- [ ] MYMACROS
- [ ] NEW MC FILE

Hint: Click on a check box to enable or disable an add-in or macro file.

***Figure 5** Customize toolbox*

Figure 6 below

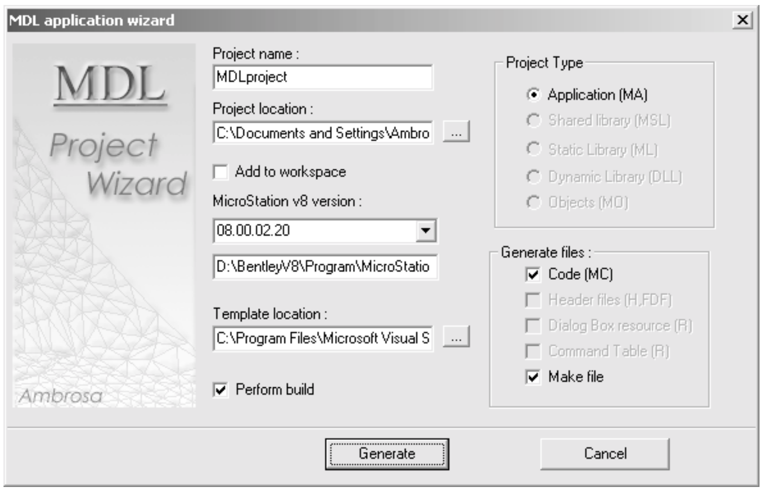
Final words

Stanislav Sumbera, Ph.D. is a LIDS and MicroStation software developer for the Berit group (www.berit.com). You may contact Stanislav Sumbera at stanislav@sumbera.com. The code may be downloaded from **www.sumbera.com**.

CAD

TIPS IN HERE

Figure 6



Project name : MDLproject
Project location : C:\Documents and Settings\Ambro
[ ] Add to workspace
MicroStation v8 version : 08.00.02.20
D:\BentleyV8\Program\MicroStatio
Template location : C:\Program Files\Microsoft Visual S
[✓] Perform build

Project Type
- [✓ Application (MA)]
- Shared library (MSL)
- Static Library (ML)
- Dynamic Library (DLL)
- Objects (MO)

Generate files :
- [✓] Code (MC)
- [ ] Header files (H,FDF)
- [ ] Dialog Box resource (R)
- [ ] Command Table (R)
- [✓] Make file

Short description of dialog items:

Project name: enter the name of you MDL project
Project location: Chose parent location for you project
Add to workspace: Check this option if you need to add newly created project into the workspace.
If not checked a new workspace will be created for your project

MicroStation version: Select from combo box target MicroStation version for your MDL application.
Template location: Only If you have different templates change the folder to the desired one
Perform build : Check-in this option if you request build after wizard is finished
Project Type: Select the MDL project type you are going to generate
Generate files: Check-in any files you want to be generated.