// This file is part of Visual D

//

// Visual D integrates the D programming language into Visual Studio

// Copyright (c) 2010-2014 by Rainer Schuetze, All Rights Reserved

//

// Distributed under the Boost Software License, Version 1.0.

// See accompanying file LICENSE\_1\_0.txt or copy at <http://www.boost.org/LICENSE_1_0.txt>

module visuald.automation;

import visuald.windows;

import std.path;

import stdext.path;

import sdk.vsi.vsshell;

import sdk.vsi.vsshell80;

import sdk.vsi.vslangproj : prjOutputTypeWinExe;

import dte = sdk.vsi.dte80a;

import visuald.comutil;

import visuald.logutil;

import visuald.dproject;

import visuald.dpackage;

import visuald.hierutil;

import visuald.chiernode;

import visuald.chiercontainer;

import visuald.pkgutil;

enum HideProjectItems = true;

class ExtProjectItem : DisposingDispatchObject, dte.ProjectItem

{

this(ExtProject prj, ExtProjectItems parent, CHierNode node)

{

mExtProject = prj;

mParent = parent;

mNode = node;

}

override void Dispose()

{

}

override HRESULT QueryInterface(in IID\* riid, void\*\* pvObject)

{

if(queryInterface!(dte.ProjectItem) (this, riid, pvObject))

return S\_OK;

return super.QueryInterface(riid, pvObject);

}

/+[id(0x0000000a), propget, hidden, helpstring("Returns value indicating whether object was changed since the last time it was saved."), helpcontext(0x0000eadb)]+/

override HRESULT get\_IsDirty(/+[out, retval]+/ VARIANT\_BOOL\* lpfReturn)

{

mixin(LogCallMix);

return returnError(E\_NOTIMPL);

}

/+[id(0x0000000a), propput, hidden, helpstring("Returns value indicating whether object was changed since the last time it was saved."), helpcontext(0x0000eadb)]+/

override HRESULT put\_IsDirty(in VARIANT\_BOOL lpfReturn)

{

mixin(LogCallMix);

return returnError(E\_NOTIMPL);

}

/+[id(0x0000000b), propget, helpstring("Returns the full pathnames of the files associated with a project item."), helpcontext(0x0000eac9)]+/

override HRESULT get\_FileNames(in short index,

/+[out, retval]+/ BSTR\* lpbstrReturn)

{

mixin(LogCallMix);

\*lpbstrReturn = allocBSTR(mNode.GetFullPath());

return S\_OK;

}

/+[id(0x0000000c), helpstring("Saves the project."), helpcontext(0x0000ea8f)]+/

override HRESULT SaveAs(in BSTR NewFileName,

/+[out, retval]+/ VARIANT\_BOOL\* lpfReturn)

{

mixin(LogCallMix);

return returnError(E\_NOTIMPL);

}

/+[id(0x0000000d), propget, helpstring("Returns the number of files associated with the project item."), helpcontext(0x0000eac4)]+/

override HRESULT get\_FileCount(/+[out, retval]+/ short\* lpsReturn)

{

\*lpsReturn = 1;

return S\_OK;

}

/+[id(00000000), propget, helpstring("Sets/returns the name of the project."), helpcontext(0x0000eae9)]+/

override HRESULT get\_Name(/+[out, retval]+/ BSTR\* pbstrReturn)

{

mixin(LogCallMix);

\*pbstrReturn = allocBSTR(mNode.GetDisplayCaption());

return S\_OK;

}

/+[id(00000000), propput, helpstring("Sets/returns the name of the project."), helpcontext(0x0000eae9)]+/

override HRESULT put\_Name(in BSTR pbstrReturn)

{

mixin(LogCallMix);

return S\_FALSE;

}

/+[id(0x00000036), propget, helpstring("Returns the collection containing the object supporting this property."), helpcontext(0x0000eab1)]+/

override HRESULT get\_Collection(/+[out, retval]+/ dte.ProjectItems \* lppcReturn)

{

mixin(LogCallMix);

\*lppcReturn = addref(mParent);

return S\_OK;

}

/+[id(0x00000038), propget, helpstring("Returns the Properties collection."), helpcontext(0x0000eaf9)]+/

override HRESULT get\_Properties(/+[out, retval]+/ dte.Properties \* ppObject)

{

mixin(LogCallMix);

return returnError(E\_NOTIMPL);

}

/+[id(0x000000c8), propget, helpstring("Returns the top-level extensibility object."), helpcontext(0x0000eac1)]+/

override HRESULT get\_DTE(/+[out, retval]+/ dte.DTE \* lppaReturn)

{

logCall("%s.get\_DTE()", this);

return GetDTE(lppaReturn);

}

/+[id(0x000000c9), get\_propget, helpstring("Returns a GUID String indicating the kind or type of the object."), helpcontext(0x0000eadd)]+/

override HRESULT get\_Kind(/+[out, retval]+/ BSTR\* lpbstrFileName)

{

mixin(LogCallMix);

return returnError(E\_NOTIMPL);

}

/+[id(0x000000cb), get\_propget, helpstring("Returns a ProjectItems collection for the object."), helpcontext(0x0000eaf6)]+/

override HRESULT get\_ProjectItems(/+[out, retval]+/ dte.ProjectItems \* lppcReturn)

{

mixin(LogCallMix);

return returnError(E\_NOTIMPL);

}

/+[id(0x000000cc), propget, helpstring("Returns value indicating whether the ProjectItem is open for a particular view."), helpcontext(0x0000eadc)]+/

override HRESULT get\_IsOpen(/+[ optional , defaultvalue("{FFFFFFFF-FFFF-FFFF-FFFF-FFFFFFFFFFFF}")]+/ in BSTR ViewKind,

/+[out, retval]+/ VARIANT\_BOOL\* lpfReturn)

{

mixin(LogCallMix);

return returnError(E\_NOTIMPL);

}

/+[id(0x000000cd), helpstring("Opens the ProjectItem object in the specified view."), helpcontext(0x0000ea88)]+/

override HRESULT Open(/+[ optional , defaultvalue("{00000000-0000-0000-0000-000000000000}")]+/ in BSTR ViewKind,

/+[out, retval]+/ dte.Window \* lppfReturn)

{

mixin(LogCallMix);

return returnError(E\_NOTIMPL);

}

/+[id(0x000000ce), helpstring("Removes an object from a collection."), helpcontext(0x0000ea8c)]+/

override HRESULT Remove()

{

mixin(LogCallMix);

return returnError(E\_NOTIMPL);

}

/+[id(0x0000006b), helpstring("Expands views of the project structure to show the ProjectItem."), helpcontext(0x0000ea7d)]+/

override HRESULT ExpandView()

{

mixin(LogCallMix);

return returnError(E\_NOTIMPL);

}

/+[id(0x0000006c), propget, helpstring("Returns an interface or object that can be accessed at run time by name."), helpcontext(0x0000ea7f)]+/

override HRESULT get\_Object(/+[out, retval]+/ IDispatch \* ProjectItemModel)

{

mixin(LogCallMix);

\*ProjectItemModel = addref(this);

return S\_OK;

}

/+[id(0x0000006d), propget, helpstring("Get an Extender for this object under the specified category."), helpcontext(0x0000eb84)]+/

override HRESULT get\_Extender(in BSTR ExtenderName,

/+[out, retval]+/ IDispatch \* Extender)

{

mixin(LogCallMix);

return returnError(E\_NOTIMPL);

}

/+[id(0x0000006e), propget, helpstring("Get a list of available Extenders on this object."), helpcontext(0x0000eb85)]+/

override HRESULT get\_ExtenderNames(/+[out, retval]+/ VARIANT\* ExtenderNames)

{

mixin(LogCallMix);

return returnError(E\_NOTIMPL);

}

/+[id(0x0000006f), propget, helpstring("Get the Extension Category ID of this object."), helpcontext(0x0000eb86)]+/

override HRESULT get\_ExtenderCATID(/+[out, retval]+/ BSTR\* pRetval)

{

mixin(LogCallMix);

return returnError(E\_NOTIMPL);

}

/+[id(0x00000071), propget, helpstring("Returns value indicating whether object was changed since the last time it was saved."), helpcontext(0x0000eadb)]+/

override HRESULT get\_Saved(/+[out, retval]+/ VARIANT\_BOOL\* lpfReturn)

{

mixin(LogCallMix);

return returnError(E\_NOTIMPL);

}

/+[id(0x00000071), propput, helpstring("Returns value indicating whether object was changed since the last time it was saved."), helpcontext(0x0000eadb)]+/

override HRESULT put\_Saved(in VARIANT\_BOOL lpfReturn)

{

mixin(LogCallMix);

return returnError(E\_NOTIMPL);

}

/+[id(0x00000074), propget, helpstring("Returns the ConfigurationManager object for this item."), helpcontext(0x0000ece9)]+/

override HRESULT get\_ConfigurationManager(/+[out, retval]+/ dte.ConfigurationManager \* ppConfigurationManager)

{

mixin(LogCallMix);

return returnError(E\_NOTIMPL);

}

/+[id(0x00000075), propget, helpstring("Returns the CodeModel object for this item."), helpcontext(0x0000ecea)]+/

override HRESULT get\_FileCodeModel(/+[out, retval]+/ dte.FileCodeModel \* ppFileCodeModel)

{

mixin(LogCallMix);

return returnError(E\_NOTIMPL);

}

/+[id(0x00000076), helpstring("Causes the item to be saved to storage."), helpcontext(0x0000ecfb)]+/

override HRESULT Save(/+[optional, defaultvalue("")]+/ BSTR FileName)

{

mixin(LogCallMix);

return returnError(E\_NOTIMPL);

}

/+[id(0x00000077), propget, helpstring("Returns the Document object for this item."), helpcontext(0x0000ecfc)]+/

override HRESULT get\_Document(/+[out, retval]+/ dte.Document \* ppDocument)

{

mixin(LogCallMix);

return returnError(E\_NOTIMPL);

}

/+[id(0x00000078), propget, helpstring("If the project item is the root of a sub-project, then returns the Project object for the sub-project."), helpcontext(0x0000ecfd)]+/

override HRESULT get\_SubProject(/+[out, retval]+/ dte.Project \* ppProject)

{

mixin(LogCallMix);

return returnError(E\_NOTIMPL);

}

/+[id(0x00000079), propget, helpstring("Returns the project that hosts this ProjectItem object."), helpcontext(0x0000ed1b)]+/

override HRESULT get\_ContainingProject(/+[out, retval]+/ dte.Project \* ppProject)

{

mixin(LogCallMix);

\*ppProject = addref(mExtProject);

return S\_OK;

}

/+[id(0x0000007a), helpstring("Removes the item from the project and it's storage."), helpcontext(0x0000ecfe)]+/

override HRESULT Delete()

{

mixin(LogCallMix);

return returnError(E\_NOTIMPL);

}

//////////////////////////////////////////////////////////////

\_\_gshared ComTypeInfoHolder mTypeHolder;

static void shared\_static\_this\_typeHolder()

{

static class \_ComTypeInfoHolder : ComTypeInfoHolder

{

override int GetIDsOfNames(

/\* [size\_is][in] \*/ in LPOLESTR \*rgszNames,

/\* [in] \*/ in UINT cNames,

/\* [size\_is][out] \*/ MEMBERID \*pMemId)

{

//mixin(LogCallMix);

if (cNames == 1 && to\_string(\*rgszNames) == "Name")

{

\*pMemId = 1;

return S\_OK;

}

return returnError(E\_NOTIMPL);

}

}

mTypeHolder = newCom!\_ComTypeInfoHolder;

addref(mTypeHolder);

}

static void shared\_static\_dtor\_typeHolder()

{

mTypeHolder = release(mTypeHolder);

}

override ComTypeInfoHolder getTypeHolder () { return mTypeHolder; }

private:

ExtProject mExtProject;

ExtProjectItems mParent;

CHierNode mNode;

};

class EmptyEnumerator : DComObject, IEnumVARIANT

{

override HRESULT QueryInterface(in IID\* riid, void\*\* pvObject)

{

if(queryInterface!(IEnumVARIANT) (this, riid, pvObject))

return S\_OK;

return super.QueryInterface(riid, pvObject);

}

HRESULT Next(in ULONG celt,

/+[out, size\_is(celt), length\_is(\*pCeltFetched)]+/ VARIANT \* rgVar,

/+[out]+/ ULONG \* pCeltFetched)

{

if(pCeltFetched)

\*pCeltFetched = 0;

return S\_FALSE;

}

HRESULT Skip(in ULONG celt)

{

return S\_OK;

}

HRESULT Reset()

{

return S\_OK;

}

HRESULT Clone(/+[out]+/ IEnumVARIANT \* ppEnum)

{

\*ppEnum = addref(this);

return S\_OK;

}

}

class ProjectItemsEnumerator : DComObject, IEnumVARIANT

{

this(ExtProjectItems item, CHierContainer node)

{

mItem = item;

mNode = node;

mCurrent = mNode.GetHead();

}

override HRESULT QueryInterface(in IID\* riid, void\*\* pvObject)

{

if(queryInterface!(IEnumVARIANT) (this, riid, pvObject))

return S\_OK;

return super.QueryInterface(riid, pvObject);

}

HRESULT Next(in ULONG celt,

/+[out, size\_is(celt), length\_is(\*pCeltFetched)]+/ VARIANT \* rgVar,

/+[out]+/ ULONG \* pCeltFetched)

{

if(!rgVar)

return E\_INVALIDARG;

ULONG c = 0;

for( ; mCurrent && c < celt; c++)

{

rgVar[c].vt = VT\_UNKNOWN;

rgVar[c].punkVal = addref(newCom!ExtProjectItem(mItem.mExtProject, mItem, mCurrent));

mCurrent = mCurrent.GetNext();

}

if(pCeltFetched)

\*pCeltFetched = c;

return c >= celt ? S\_OK : S\_FALSE;

}

HRESULT Skip(in ULONG celt)

{

foreach(\_; 0 .. celt)

{

if(!mCurrent)

return S\_FALSE;

mCurrent = mCurrent.GetNext();

}

return S\_OK;

}

HRESULT Reset()

{

mCurrent = mNode.GetHead();

return S\_OK;

}

HRESULT Clone(/+[out]+/ IEnumVARIANT \* ppEnum)

{

\*ppEnum = addref(newCom!ProjectItemsEnumerator(mItem, mNode));

return S\_OK;

}

ExtProjectItems mItem;

CHierContainer mNode;

CHierNode mCurrent;

}

class ProjectRootEnumerator : DComObject, IEnumVARIANT

{

this(ExtProject prj)

{

mProject = prj;

mDone = false;

}

override HRESULT QueryInterface(in IID\* riid, void\*\* pvObject)

{

if(queryInterface!(IEnumVARIANT) (this, riid, pvObject))

return S\_OK;

return super.QueryInterface(riid, pvObject);

}

HRESULT Next(in ULONG celt,

/+[out, size\_is(celt), length\_is(\*pCeltFetched)]+/ VARIANT \* rgVar,

/+[out]+/ ULONG \* pCeltFetched)

{

if(!rgVar)

return E\_INVALIDARG;

ULONG fetched = 0;

if(celt > 0 && !mDone)

{

rgVar.vt = VT\_UNKNOWN;

rgVar.punkVal = addref(mProject);

mDone = true;

fetched = 1;

}

if(pCeltFetched)

\*pCeltFetched = fetched;

return fetched >= celt ? S\_OK : S\_FALSE;

}

HRESULT Skip(in ULONG celt)

{

if(celt > 0)

mDone = true;

return !mDone ? S\_OK : S\_FALSE;

}

HRESULT Reset()

{

mDone = false;

return S\_OK;

}

HRESULT Clone(/+[out]+/ IEnumVARIANT \* ppEnum)

{

\*ppEnum = addref(newCom!ProjectRootEnumerator(mProject));

return S\_OK;

}

ExtProject mProject;

bool mDone;

}

class ExtProjectItems : DisposingDispatchObject, dte.ProjectItems

{

this(ExtProject prj, ExtProjectItems parent, CHierNode node)

{

mExtProject = prj;

mParent = parent;

mNode = node;

}

\_\_gshared ComTypeInfoHolder mTypeHolder;

override ComTypeInfoHolder getTypeHolder () { return mTypeHolder; }

override void Dispose()

{

}

override HRESULT QueryInterface(in IID\* riid, void\*\* pvObject)

{

//mixin(LogCallMix);

if(queryInterface!(dte.ProjectItems) (this, riid, pvObject))

return S\_OK;

return super.QueryInterface(riid, pvObject);

}

override int Item(

/\* [in] \*/ in VARIANT index,

/\* [retval][out] \*/ dte.ProjectItem \*lppcReturn)

{

mixin(LogCallMix);

return returnError(E\_NOTIMPL);

}

override int get\_Parent(

/\* [retval][out] \*/ IDispatch\* lppptReturn)

{

mixin(LogCallMix);

\*lppptReturn = addref(mParent);

return S\_OK;

}

override int get\_Count(

/\* [retval][out] \*/ int \*lplReturn)

{

logCall("%s.get\_Count(lplReturn=%s)", this, lplReturn);

static if(HideProjectItems)

\*lplReturn = 0;

else if(auto c = cast(CHierContainer) mNode)

\*lplReturn = c.GetCount();

else

\*lplReturn = 0;

return S\_OK;

}

override int \_NewEnum(

/\* [retval][out] \*/ IUnknown \*lppiuReturn)

{

mixin(LogCallMix);

static if(HideProjectItems)

\*lppiuReturn = addref(newCom!EmptyEnumerator());

else if(auto c = cast(CHierContainer) mNode)

\*lppiuReturn = addref(newCom!ProjectItemsEnumerator(this, c));

else

\*lppiuReturn = addref(newCom!EmptyEnumerator());

return S\_OK;

}

override int get\_DTE(

/\* [retval][out] \*/ dte.DTE        \*lppaReturn)

{

logCall("%s.get\_DTE()", this);

return GetDTE(lppaReturn);

}

override int get\_Kind(

/\* [retval][out] \*/ BSTR \*lpbstrFileName)

{

logCall("%s.get\_Kind(lpbstrFileName=%s)", this, lpbstrFileName);

return returnError(E\_NOTIMPL);

}

override int AddFromFile(

/\* [in] \*/ in BSTR FileName,

/\* [retval][out] \*/ dte.ProjectItem \*lppcReturn)

{

mixin(LogCallMix);

return returnError(E\_NOTIMPL);

}

override int AddFromTemplate(

/\* [in] \*/ in BSTR FileName,

/\* [in] \*/ in BSTR Name,

/\* [retval][out] \*/ dte.ProjectItem \*lppcReturn)

{

mixin(LogCallMix);

return returnError(E\_NOTIMPL);

}

override int AddFromDirectory(

/\* [in] \*/ in BSTR Directory,

/\* [retval][out] \*/ dte.ProjectItem \*lppcReturn)

{

logCall("AddFromDirectory(Directory=%s, lppcReturn=%s)", \_toLog(Directory), \_toLog(lppcReturn));

return returnError(E\_NOTIMPL);

}

override int get\_ContainingProject(

/\* [retval][out] \*/ dte.Project\* ppProject)

{

mixin(LogCallMix);

\*ppProject = addref(mExtProject);

return S\_OK;

}

override int AddFolder(

BSTR Name,

/\* [defaultvalue] \*/ BSTR Kind,

/\* [retval][out] \*/ dte.ProjectItem \*pProjectItem)

{

logCall("AddFolder(Kind=%s, pProjectItem=%s)", \_toLog(Kind), \_toLog(pProjectItem));

return returnError(E\_NOTIMPL);

}

override int AddFromFileCopy(

BSTR FilePath,

/\* [retval][out] \*/ dte.ProjectItem \*pProjectItem)

{

logCall("AddFromFileCopy(FilePath=%s, pProjectItem=%s)", \_toLog(FilePath), \_toLog(pProjectItem));

return returnError(E\_NOTIMPL);

}

override int Invoke(/\* [in] \*/ in DISPID dispIdMember,

/\* [in] \*/ in IID\* riid,

/\* [in] \*/ in LCID lcid,

/\* [in] \*/ in WORD wFlags,

/\* [out][in] \*/ DISPPARAMS \*pDispParams,

/\* [out] \*/ VARIANT \*pVarResult,

/\* [out] \*/ EXCEPINFO \*pExcepInfo,

/\* [out] \*/ UINT \*puArgErr)

{

mixin(LogCallMix);

if (dispIdMember == -4)

{

pVarResult.vt = VT\_UNKNOWN;

return \_NewEnum(&pVarResult.punkVal);

}

return super.Invoke(dispIdMember, riid, lcid, wFlags, pDispParams, pVarResult, pExcepInfo, puArgErr);

}

ExtProject mExtProject;

ExtProjectItems mParent;

CHierNode mNode;

};

class ExtProjectRootItems : ExtProjectItems

{

this(ExtProject prj, ExtProjectItems parent, CHierNode node)

{

super(prj, parent, node);

}

override int \_NewEnum(/\* [retval][out] \*/ IUnknown \*lppiuReturn)

{

mixin(LogCallMix);

\*lppiuReturn = addref(newCom!ProjectRootEnumerator(mExtProject));

return S\_OK;

}

}

class ExtProperties : DisposingDispatchObject, dte.Properties

{

this(ExtProject prj)

{

mProject = prj;

}

override void Dispose()

{

}

override HRESULT QueryInterface(in IID\* riid, void\*\* pvObject)

{

//mixin(LogCallMix);

if(queryInterface!(dte.Properties) (this, riid, pvObject))

return S\_OK;

return super.QueryInterface(riid, pvObject);

}

override HRESULT Item(in VARIANT index, dte.Property \* lplppReturn)

{

mixin(LogCallMix);

if(index.vt != VT\_BSTR)

return E\_INVALIDARG;

string prop = to\_string(index.bstrVal);

if(prop == "FullPath")

{

string fullpath = mProject.mProject.GetFilename();

\*lplppReturn = addref(newCom!(ExtProperty!string)(this, prop, fullpath));

return S\_OK;

}

if(prop == "ProjectDirectory")

{

string fullpath = dirName(mProject.mProject.GetFilename());

\*lplppReturn = addref(newCom!(ExtProperty!string)(this, prop, fullpath));

return S\_OK;

}

return returnError(S\_FALSE);

}

/+[id(0x00000001), propget, restricted, hidden]+/

override HRESULT get\_Application(/+[out, retval]+/ IDispatch \* lppidReturn)

{

mixin(LogCallMix);

return returnError(E\_NOTIMPL);

}

/+[id(0x00000002), propget, helpstring("Returns the parent object."), helpcontext(0x0000eaf2)]+/

override HRESULT get\_Parent(/+[out, retval]+/ IDispatch \* lppidReturn)

{

mixin(LogCallMix);

return returnError(E\_NOTIMPL);

}

/+[id(0x00000028), propget, helpstring("Returns value indicating the count of objects in the collection."), helpcontext(0x0000eabb)]+/

override HRESULT get\_Count(/+[out, retval]+/ int\* lplReturn)

{

mixin(LogCallMix);

return returnError(E\_NOTIMPL);

}

/+[id(0xfffffffc), restricted]+/

override HRESULT \_NewEnum(/+[out, retval]+/ IUnknown \* lppiuReturn)

{

mixin(LogCallMix);

return returnError(E\_NOTIMPL);

}

/+[id(0x00000064), propget, helpstring("Returns the top-level extensibility object."), helpcontext(0x0000eac1)]+/

override HRESULT get\_DTE(/+[out, retval]+/ dte.DTE \* lppaReturn)

{

logCall("%s.get\_DTE()", this);

return GetDTE(lppaReturn);

}

//////////////////////////////////////////////////////////////

\_\_gshared ComTypeInfoHolder mTypeHolder;

static void shared\_static\_this\_typeHolder()

{

static class \_ComTypeInfoHolder : ComTypeInfoHolder

{

override int GetIDsOfNames(

/\* [size\_is][in] \*/ in LPOLESTR \*rgszNames,

/\* [in] \*/ in UINT cNames,

/\* [size\_is][out] \*/ MEMBERID \*pMemId)

{

//mixin(LogCallMix);

if (cNames == 1 && to\_string(\*rgszNames) == "Name")

{

\*pMemId = 1;

return S\_OK;

}

return returnError(E\_NOTIMPL);

}

}

mTypeHolder = newCom!\_ComTypeInfoHolder;

addref(mTypeHolder);

}

static void shared\_static\_dtor\_typeHolder()

{

mTypeHolder = release(mTypeHolder);

}

override ComTypeInfoHolder getTypeHolder () { return mTypeHolder; }

private:

ExtProject mProject;

}

class ExtProperty(T) : DisposingDispatchObject, dte.Property

{

this(ExtProperties props, string name, T value)

{

mProperties = props;

mName = name;

mValue = value;

}

override void Dispose()

{

}

override HRESULT QueryInterface(in IID\* riid, void\*\* pvObject)

{

//mixin(LogCallMix);

if(queryInterface!(dte.Property) (this, riid, pvObject))

return S\_OK;

return super.QueryInterface(riid, pvObject);

}

override HRESULT get\_Value(/+[out, retval]+/ VARIANT\* lppvReturn)

{

mixin(LogCallMix);

static if(is (T == string))

{

lppvReturn.vt = VT\_BSTR;

lppvReturn.bstrVal = allocBSTR(mValue);

}

else static if (is(T == int))

{

lppvReturn.vt = VT\_INT;

lppvReturn.intVal = mValue;

}

else

static assert(false, "unsupported type in ExtProperty.Value");

return S\_OK;

}

/+[id(00000000), propput, helpstring("Sets/ returns the value of property returned by the Property object."), helpcontext(0x0000eb08)]+/

override HRESULT put\_Value(in VARIANT lppvSet)

{

mixin(LogCallMix);

return returnError(S\_FALSE);

}

/+[id(00000000), propputref, helpstring("Sets/ returns the value of property returned by the Property object."), helpcontext(0x0000eb08)]+/

override HRESULT putref\_Value(in VARIANT lppvReturn)

{

mixin(LogCallMix);

return returnError(S\_FALSE);

}

/+[id(0x00000003), propget, helpstring("Returns one element of a list."), helpcontext(0x0000ead6)]+/

override HRESULT get\_IndexedValue(in VARIANT Index1,

/+[ optional]+/ in VARIANT Index2,

/+[ optional]+/ in VARIANT Index3,

/+[ optional]+/ in VARIANT Index4,

/+[out, retval]+/ VARIANT\* Val)

{

mixin(LogCallMix);

return returnError(E\_NOTIMPL);

}

/+[id(0x00000003), propput, helpstring("Returns one element of a list."), helpcontext(0x0000ead6)]+/

override HRESULT put\_IndexedValue(in VARIANT Index1,

/+[ optional]+/ in VARIANT Index2,

/+[ optional]+/ in VARIANT Index3,

/+[ optional]+/ in VARIANT Index4,

in VARIANT Val)

{

mixin(LogCallMix);

return returnError(E\_NOTIMPL);

}

/+[id(0x00000004), propget, helpstring("Returns a value representing the number of items in the list value."), helpcontext(0x0000eaea)]+/

override HRESULT get\_NumIndices(/+[out, retval]+/ short\* lpiRetVal)

{

mixin(LogCallMix);

return returnError(E\_NOTIMPL);

}

/+[id(0x00000001), propget, restricted, hidden]+/

override HRESULT get\_Application(/+[out, retval]+/ IDispatch \* lppidReturn)

{

mixin(LogCallMix);

return returnError(E\_NOTIMPL);

}

/+[id(0x00000002), propget, restricted, hidden]+/

override HRESULT get\_Parent(/+[out, retval]+/ dte.Properties \* lpppReturn)

{

mixin(LogCallMix);

\*lpppReturn = addref(mProperties);

return S\_OK;

}

/+[id(0x00000028), propget, helpstring("Returns the name of the object."), helpcontext(0x0000edbb)]+/

override HRESULT get\_Name(/+[out, retval]+/ BSTR\* lpbstrReturn)

{

mixin(LogCallMix);

\*lpbstrReturn = allocBSTR(mName);

return S\_OK;

}

/+[id(0x0000002a), propget, helpstring("Returns the collection containing the object supporting this property."), helpcontext(0x0000eab1)]+/

override HRESULT get\_Collection(/+[out, retval]+/ dte.Properties \* lpppReturn)

{

mixin(LogCallMix);

\*lpppReturn = addref(mProperties);

return S\_OK;

}

/+[id(0x0000002d), propget, helpstring("Sets/returns value of Property object when type of value is Object."), helpcontext(0x0000eaed)]+/

override HRESULT get\_Object(/+[out, retval]+/ IDispatch \* lppunk)

{

mixin(LogCallMix);

return returnError(E\_NOTIMPL);

}

/+[id(0x0000002d), propputref, helpstring("Sets/returns value of Property object when type of value is Object."), helpcontext(0x0000eaed)]+/

override HRESULT putref\_Object(/+[in]+/ IUnknown lppunk)

{

mixin(LogCallMix);

return returnError(E\_NOTIMPL);

}

/+[id(0x00000064), propget, helpstring("Returns the top-level extensibility object."), helpcontext(0x0000eac1)]+/

override HRESULT get\_DTE(/+[out, retval]+/ dte.DTE \* lppaReturn)

{

logCall("%s.get\_DTE()", this);

return GetDTE(lppaReturn);

}

//////////////////////////////////////////////////////////////

\_\_gshared ComTypeInfoHolder mTypeHolder;

static void shared\_static\_this\_typeHolder()

{

static class \_ComTypeInfoHolder : ComTypeInfoHolder

{

override int GetIDsOfNames(

/\* [size\_is][in] \*/ in LPOLESTR \*rgszNames,

/\* [in] \*/ in UINT cNames,

/\* [size\_is][out] \*/ MEMBERID \*pMemId)

{

//mixin(LogCallMix);

if (cNames == 1 && to\_string(\*rgszNames) == "Name")

{

\*pMemId = 1;

return S\_OK;

}

return returnError(E\_NOTIMPL);

}

}

mTypeHolder = newCom!\_ComTypeInfoHolder;

addref(mTypeHolder);

}

static void shared\_static\_dtor\_typeHolder()

{

mTypeHolder = release(mTypeHolder);

}

override ComTypeInfoHolder getTypeHolder () { return mTypeHolder; }

private:

string mName;

T mValue;

ExtProperties mProperties;

}

class ExtProject : ExtProjectItem, dte.Project

{

this(Project prj)

{

super(this, null, prj.GetProjectNode());

mProject = prj;

mProperties = newCom!ExtProperties(this);

}

override HRESULT QueryInterface(in IID\* riid, void\*\* pvObject)

{

//mixin(LogCallMix);

if(queryInterface!(dte.Project) (this, riid, pvObject))

return S\_OK;

return super.QueryInterface(riid, pvObject);

}

// DTE.Project

override int get\_Name(

/\* [retval][out] \*/ BSTR \*lpbstrName)

{

logCall("%s.get\_Name(lpbstrName=%s)", this, \_toLog(lpbstrName));

\*lpbstrName = allocBSTR(mProject.GetCaption());

return S\_OK;

}

override int put\_Name(

/\* [in] \*/ in BSTR bstrName)

{

logCall("%s.put\_Name(bstrName=%s)", this, \_toLog(bstrName));

mProject.SetCaption(to\_string(bstrName));

return S\_OK;

}

override int get\_FileName(

/\* [retval][out] \*/ BSTR \*lpbstrName)

{

logCall("%s.get\_FileName(lpbstrName=%s)", this, \_toLog(lpbstrName));

\*lpbstrName = allocBSTR(mProject.GetFilename());

return S\_OK;

}

override int get\_IsDirty(

/\* [retval][out] \*/ VARIANT\_BOOL \*lpfReturn)

{

mixin(LogCallMix);

return returnError(E\_NOTIMPL);

}

override int put\_IsDirty(

/\* [in] \*/ in VARIANT\_BOOL Dirty)

{

logCall("%s.put\_IsDirty(Dirty=%s)", this, \_toLog(Dirty));

return returnError(E\_NOTIMPL);

}

override int get\_Collection(

/\* [retval][out] \*/ dte.Projects \*lppaReturn)

{

mixin(LogCallMix);

dte2.DTE2 \_dte = GetDTE();

if(!\_dte)

return returnError(E\_FAIL);

scope(exit) release(\_dte);

IUnknown solution; // dte.Solution not derived from IUnknown?!

if(\_dte.get\_Solution(cast(dte.Solution\*)&solution) != S\_OK || !solution)

return returnError(E\_FAIL);

scope(exit) release(solution);

dte.\_Solution \_solution = qi\_cast!(dte.\_Solution)(solution);

if(!\_solution)

return returnError(E\_FAIL);

scope(exit) release(\_solution);

return \_solution.get\_Projects(lppaReturn);

}

override int SaveAs(

/\* [in] \*/ in BSTR NewFileName)

{

logCall("%s.SaveAs(NewFileName=%s)", this, \_toLog(NewFileName));

return returnError(E\_NOTIMPL);

}

override int get\_DTE(

/\* [retval][out] \*/ dte.DTE        \*lppaReturn)

{

logCall("%s.get\_DTE()", this);

return GetDTE(lppaReturn);

}

override int get\_Kind(

/\* [retval][out] \*/ BSTR \*lpbstrName)

{

logCall("%s.get\_Kind(lpbstrName=%s)", this, \_toLog(lpbstrName));

wstring s = GUID2wstring(g\_projectFactoryCLSID);

\*lpbstrName = allocwBSTR(s);

return S\_OK;

}

override int get\_ProjectItems(

/\* [retval][out] \*/ dte.ProjectItems\* lppcReturn)

{

mixin(LogCallMix);

\*lppcReturn = addref(newCom!ExtProjectItems(this, null, mProject.GetProjectNode()));

return S\_OK;

}

override int get\_Properties(

/\* [retval][out] \*/ dte.Properties \*ppObject)

{

mixin(LogCallMix);

\*ppObject = addref(mProperties);

return S\_OK;

}

override int get\_UniqueName(

/\* [retval][out] \*/ BSTR \*lpbstrName)

{

logCall("%s.get\_UniqueName(lpbstrName=%s)", this, \_toLog(lpbstrName));

if (!mProject)

return returnError(E\_FAIL);

IVsSolution srpSolution = queryService!(IVsSolution);

if(!srpSolution)

return returnError(E\_FAIL);

IVsHierarchy pIVsHierarchy = mProject; // ->GetIVsHierarchy();

int hr = srpSolution.GetUniqueNameOfProject(pIVsHierarchy, lpbstrName);

srpSolution.Release();

return hr;

}

override int get\_Object(

/\* [retval][out] \*/ IDispatch\* ProjectModel)

{

logCall("%s.get\_Object(out ProjectModel=%s)", this, \_toLog(&ProjectModel));

\*ProjectModel = addref(this); // (mProject);

return S\_OK;

}

override int get\_Extender(

/\* [in] \*/ in BSTR ExtenderName,

/\* [retval][out] \*/ IDispatch \*Extender)

{

logCall("%s.get\_Extender(ExtenderName=%s)", this, \_toLog(ExtenderName));

return returnError(E\_NOTIMPL);

}

override int get\_ExtenderNames(

/\* [retval][out] \*/ VARIANT \*ExtenderNames)

{

logCall("%s.get\_ExtenderNames(ExtenderNames=%s)", this, \_toLog(ExtenderNames));

return returnError(E\_NOTIMPL);

}

override int get\_ExtenderCATID(

/\* [retval][out] \*/ BSTR \*pRetval)

{

logCall("%s.get\_ExtenderCATID(pRetval=%s)", this, \_toLog(pRetval));

return returnError(E\_NOTIMPL);

}

override int get\_FullName(

/\* [retval][out] \*/ BSTR \*lpbstrName)

{

logCall("%s.get\_FullName(lpbstrName=%s)", this, \_toLog(lpbstrName));

return get\_FileName(lpbstrName);

}

override int get\_Saved(

/\* [retval][out] \*/ VARIANT\_BOOL \*lpfReturn)

{

mixin(LogCallMix);

return returnError(E\_NOTIMPL);

}

override int put\_Saved(

/\* [in] \*/ in VARIANT\_BOOL SavedFlag)

{

logCall("put\_Saved(SavedFlag=%s)", \_toLog(SavedFlag));

return returnError(E\_NOTIMPL);

}

override int get\_ConfigurationManager(

/\* [retval][out] \*/ dte.ConfigurationManager\* ppConfigurationManager)

{

mixin(LogCallMix);

\*ppConfigurationManager = mProject.getConfigurationManager();

return S\_OK;

}

override int get\_Globals(

/\* [retval][out] \*/ dte.Globals\* ppGlobals)

{

mixin(LogCallMix);

HRESULT hr = S\_OK;

// hr = CheckEnabledItem(this, &IID\_\_DTE, L"Globals");

// IfFailRet(hr);

// if don't already have m\_srpGlobals, get it from shell

IVsExtensibility3 ext = queryService!(dte.IVsExtensibility, IVsExtensibility3);

if(!ext)

return E\_FAIL;

scope(exit) release(ext);

dte.Globals globals;

VARIANT varIVsGlobalsCallback;

varIVsGlobalsCallback.vt = VT\_UNKNOWN;

IVsHierarchy pIVsHierarchy = mProject;

hr = mProject.QueryInterface(&IID\_IUnknown, cast(void\*\*)&varIVsGlobalsCallback.punkVal);

if(!FAILED(hr))

{

//! TODO fix: returns failure

hr = ext.GetGlobalsObject(varIVsGlobalsCallback, cast(IUnknown\*) &globals);

varIVsGlobalsCallback.punkVal.Release();

}

\*ppGlobals = globals;

return hr;

}

override int Save(

/\* [defaultvalue] \*/ BSTR FileName)

{

logCall("Save(FileName=%s)", \_toLog(FileName));

return returnError(E\_NOTIMPL);

}

///////////////////////////////////////////

extern(D)

static bool searchNestedHierarchy(IVsHierarchy pHierarchy, VSITEMID item,

scope bool delegate (IVsHierarchy, VSITEMID, IVsHierarchy, VSITEMID) dg)

{

VARIANT var;

if((pHierarchy.GetProperty(item, VSHPROPID\_Container, &var) == S\_OK &&

((var.vt == VT\_BOOL && var.boolVal) || (var.vt == VT\_I4 && var.lVal))) ||

(pHierarchy.GetProperty(item, VSHPROPID\_Expandable, &var) == S\_OK &&

((var.vt == VT\_BOOL && var.boolVal) || (var.vt == VT\_I4 && var.lVal))))

{

IVsHierarchy nestedHierarchy;

VSITEMID itemidNested;

if(pHierarchy.GetNestedHierarchy(item, &IVsHierarchy.iid, cast(void \*\*)&nestedHierarchy, &itemidNested) == S\_OK)

{

scope(exit) release(nestedHierarchy);

if(dg(pHierarchy, item, nestedHierarchy, itemidNested))

return true;

if(searchNestedHierarchy(nestedHierarchy, itemidNested, dg))

return true;

}

else if(pHierarchy.GetProperty(item, VSHPROPID\_FirstChild, &var) == S\_OK &&

(var.vt == VT\_INT\_PTR || var.vt == VT\_I4 || var.vt == VT\_INT))

{

VSITEMID chid = var.lVal;

while(chid != VSITEMID\_NIL)

{

VARIANT name;

pHierarchy.GetProperty(item, VSHPROPID\_Name, &name);

detachBSTR(name.bstrVal);

if(searchNestedHierarchy(pHierarchy, chid, dg))

return true;

if(pHierarchy.GetProperty(chid, VSHPROPID\_NextSibling, &var) != S\_OK ||

(var.vt != VT\_INT\_PTR && var.vt != VT\_I4 && var.vt != VT\_INT))

break;

chid = var.lVal;

}

}

}

return false;

}

override int get\_ParentProjectItem(

/\* [retval][out] \*/ dte.ProjectItem \*ppParentProjectItem)

{

mixin(LogCallMix);

\*ppParentProjectItem = null;

IVsSolution srpSolution = queryService!(IVsSolution);

if(!srpSolution)

return returnError(E\_FAIL);

int hr = E\_UNEXPECTED;

IVsHierarchy pIVsHierarchy = mProject; // ->GetIVsHierarchy();

auto hierSolution = qi\_cast!(IVsHierarchy)(srpSolution);

if (hierSolution)

{

IVsHierarchy parentHier;

VSITEMID parentItem;

bool matchItem(IVsHierarchy hier, VSITEMID item, IVsHierarchy nestedHier, VSITEMID nestItem)

{

if (nestedHier == pIVsHierarchy)

{

parentHier = addref(hier);

parentItem = item;

return true;

}

return false;

}

if (searchNestedHierarchy(hierSolution, VSITEMID\_ROOT, &matchItem))

{

VARIANT var;

hr = parentHier.GetProperty(VSITEMID\_ROOT, VSHPROPID\_ExtObject, &var);

if (hr == S\_OK && var.vt == VT\_DISPATCH && var.pdispVal)

{

\*ppParentProjectItem = qi\_cast!(dte.ProjectItem)(var.pdispVal);

release(var.pdispVal);

}

else if (hr == S\_OK)

hr = E\_UNEXPECTED;

}

release(parentHier);

release(hierSolution);

}

release(srpSolution);

return hr;

}

override int get\_CodeModel(

/\* [retval][out] \*/ dte.CodeModel \*ppCodeModel)

{

mixin(LogCallMix);

\*ppCodeModel = null;

return S\_OK; // returnError(E\_NOTIMPL);

}

override int Delete()

{

mixin(LogCallMix);

return returnError(E\_NOTIMPL);

}

//////////////////////////////////////////////////////////////

\_\_gshared ComTypeInfoHolder mTypeHolder;

static void shared\_static\_this\_typeHolder()

{

static class \_ComTypeInfoHolder : ComTypeInfoHolder

{

override int GetIDsOfNames(

/\* [size\_is][in] \*/ in LPOLESTR \*rgszNames,

/\* [in] \*/ in UINT cNames,

/\* [size\_is][out] \*/ MEMBERID \*pMemId)

{

//mixin(LogCallMix);

if (cNames == 1 && to\_string(\*rgszNames) == "Name")

{

\*pMemId = 1;

return S\_OK;

}

return returnError(E\_NOTIMPL);

}

}

mTypeHolder = newCom!\_ComTypeInfoHolder;

addref(mTypeHolder);

}

static void shared\_static\_dtor\_typeHolder()

{

mTypeHolder = release(mTypeHolder);

}

override ComTypeInfoHolder getTypeHolder () { return mTypeHolder; }

Project mProject;

dte.Properties mProperties;

}

void automation\_shared\_static\_this\_typeHolder()

{

ExtProjectItem.shared\_static\_this\_typeHolder();

ExtProperties.shared\_static\_this\_typeHolder();

ExtProperty!string.shared\_static\_this\_typeHolder();

ExtProperty!int.shared\_static\_this\_typeHolder();

ExtProject.shared\_static\_this\_typeHolder();

}

void automation\_shared\_static\_dtor\_typeHolder()

{

ExtProjectItem.shared\_static\_dtor\_typeHolder();

ExtProperties.shared\_static\_dtor\_typeHolder();

ExtProperty!string.shared\_static\_dtor\_typeHolder();

ExtProperty!int.shared\_static\_dtor\_typeHolder();

ExtProject.shared\_static\_dtor\_typeHolder();

}

// This file is part of Visual D

//

// Visual D integrates the D programming language into Visual Studio

// Copyright (c) 2010 by Rainer Schuetze, All Rights Reserved

//

// Distributed under the Boost Software License, Version 1.0.

// See accompanying file LICENSE\_1\_0.txt or copy at <http://www.boost.org/LICENSE_1_0.txt>

module visuald.chiercontainer;

import visuald.windows;

import std.string;

import std.path;

import std.utf;

import sdk.vsi.vsshell;

import visuald.hierarchy;

import visuald.chiernode;

import visuald.hierutil;

import visuald.comutil;

//-----------------------------------------------------------------------------

// Name: CHierContainer

//

// Description:

// Class for every object in a hierarchy that has children. Implements the

// idea of a node that has children, relies on CHierNode to take care of

// parent/sibling info.

//

//-----------------------------------------------------------------------------

class CHierContainer : CHierNode

{

~this()

{

//DeleteAll(null);

}

override void removeFromItemMap(bool recurse)

{

if(recurse)

{

for(CHierNode n = GetHeadEx(false); n; n = n.GetNext(false))

n.removeFromItemMap(recurse);

}

super.removeFromItemMap(recurse);

}

public:

// CHierNode overrides

override bool Expandable() { return GetHead(true) !is null; }

override bool ExpandByDefault() { return false; }

override bool IsContainer() { return true; }

override int OnStartLabelEdit()

{

return S\_OK;

}

override int OnCommitLabelEdit()

{

return S\_OK;

}

override int OnCancelLabelEdit()

{

return S\_OK;

}

override VSITEMID GetFirstChildID(bool fDisplayOnly = true)

{

CHierNode head = GetHeadEx(fDisplayOnly);

return head ? head.GetVsItemID() : VSITEMID\_NIL;

}

// Used by the hierarchy in response to VSHPROPID\_FirstChild/VSHPROPID\_NextSibling. These

// properties are spec'd to only return member items (visible or not)

override VSITEMID GetFirstMemberChildID()

{

CHierNode pNode = GetHeadEx(false);

while(pNode && !pNode.IsMemberItem())

pNode = pNode.GetNext(false);

return pNode ? pNode.GetVsItemID() : VSITEMID\_NIL;

}

// CHierContainer methods

int Refresh(CVsHierarchy pCVsHierarchy)

{

DeleteAll(pCVsHierarchy);

SetChildrenBeenEnumerated(false);

return S\_OK;

}

int EnumerateChildren() { return S\_OK; }

override CHierNode GetHeadEx(bool fDisplayOnly = true)

{

if (!HaveChildrenBeenEnumerated())

{

// CWaitCursor cursWait;

HRESULT hr = EnumerateChildren();

SetChildrenBeenEnumerated(true);

if(FAILED(hr))

// Failed to enumerate children. Just return that we don't have any.

return null;

}

return GetHead(fDisplayOnly);

}

CHierNode GetHead(bool fDisplayOnly = true)

{

if (!fDisplayOnly)

return m\_pHeadNode;

CHierNode pNode = m\_pHeadNode;

if (pNode && !pNode.IsDisplayable())

pNode = pNode.GetNext();

return pNode;

}

CHierNode GetTail()

{

return m\_pTailNode;

}

int GetCount(bool fDisplayOnly = true) // return number of children

{

int n = 0;

CHierNode pNext = GetHead(fDisplayOnly);

while (pNext)

{

pNext = pNext.GetNext(fDisplayOnly);

++n;

}

return n;

}

CHierNode GetPrevChildOf(CHierNode pCurrent, bool fDisplayOnly = true)

{

assert(m\_pHeadNode);

if (pCurrent is m\_pHeadNode)

return null;

CHierNode pNodePrev = m\_pHeadNode;

while (pNodePrev && pNodePrev.GetNext(fDisplayOnly) !is pCurrent)

pNodePrev = pNodePrev.GetNext(fDisplayOnly);

// If the node we end up with isn't displayable, then there are not

// any displayble nodes...

if (pNodePrev && (fDisplayOnly && !pNodePrev.IsDisplayable()))

pNodePrev = null;

return pNodePrev;

}

// Override to get custom add behavior such as keeping the list sorted.

// If not sorted list, it calls AddTail(), else calls AddSorted();

void Add(CHierNode pNode)

{

if(IsSortedList())

AddSorted(null, pNode);

else

AddTail(pNode);

}

void AddAfter(CHierNode pCurrNode, CHierNode pNewNode)

{

if(IsSortedList())

{

AddSorted(pCurrNode, pNewNode);

}

else if (pCurrNode)

{

pNewNode.SetNext(pCurrNode.GetNext(false));

pNewNode.SetParent(pCurrNode.GetParent());

pCurrNode.SetNext(pNewNode);

if (pCurrNode is m\_pTailNode)

m\_pTailNode = pNewNode;

// Finally, inform the hierarchy.

NotifyHierarchyOfAdd(pNewNode);

}

else

{

AddHead(pNewNode);

}

}

void AddHead(CHierNode pNode)

{

assert(pNode);

pNode.SetParent(this);

pNode.SetNext(m\_pHeadNode);

m\_pHeadNode = pNode;

if (!m\_pTailNode)

m\_pTailNode = pNode;

NotifyHierarchyOfAdd(pNode);

//                addref(pNode);

}

void AddTail(CHierNode pNode)

{

assert(pNode);

pNode.SetParent(this);

pNode.SetNext(null);

if(m\_pTailNode)

{

assert(m\_pHeadNode);

m\_pTailNode.SetNext(pNode);

m\_pTailNode = pNode;

}

else

{

assert(!m\_pHeadNode);

m\_pHeadNode = m\_pTailNode = pNode;

}

NotifyHierarchyOfAdd(pNode);

//                addref(pNode);

}

HRESULT Remove(CHierNode pNode)

{

assert(pNode);

CHierNode pNodeCur = m\_pHeadNode; // The node to be removed

CHierNode pNodePrev = null; // fix this node's next pointer

while (pNode !is pNodeCur && pNodeCur)

{ // find pNode in list of children

pNodePrev = pNodeCur;

pNodeCur = pNodeCur.GetNext(false);

}

// ASSERT if caller gave a node not in the list

assert(pNodeCur);

if (!pNodeCur)

return E\_FAIL;

// Then we found the node in the list. (this is a good thing!)

if (pNodeCur is m\_pHeadNode)

{ // pNode is the HeadNode

assert(pNode is m\_pHeadNode);

m\_pHeadNode = pNode.GetNext(false);

if (!m\_pHeadNode)

{ // single child case

m\_pTailNode = null;

}

}

else if (pNodeCur is m\_pTailNode)

{ // We are removing the last node.

m\_pTailNode = pNodePrev;

pNodePrev.SetNext(null);

}

else

{ // We are just removing a node in the middle.

pNodePrev.SetNext(pNode.GetNext(false));

}

pNode.SetParent(null);

pNode.SetNext(null);

//                release(pNode);

pNode.removeFromItemMap(true);

return S\_OK;

}

HRESULT Delete(CHierNode pNode, CVsHierarchy pCVsHierarchy)

{

if (!pNode)

return E\_INVALIDARG;

HRESULT hr;

if (pCVsHierarchy)

{

hr = pCVsHierarchy.OnItemDeleted(pNode);

assert(SUCCEEDED(hr));

}

return Remove(pNode);

}

void DeleteAll(CVsHierarchy pCVsHierarchy)

{

while (GetHead(false))

{

HRESULT hr = Delete(GetHead(false), pCVsHierarchy);

assert(SUCCEEDED(hr));

}

}

/+

HRESULT CloseDocuments(bool bPromptToSave = FALSE);

// returns the node from child list who's GetDisplayName() == pszName

CHierNode GetNodeByName(LPCTSTR pszName, bool fDisplayOnly = TRUE);

CHierNode GetNodeByIndex(DWORD dwIndex, bool fDisplayOnly = TRUE);

CHierNode GetNodeByVariant(VARIANT \*pvar, bool fDisplayOnly = TRUE);

+/

// Allows walking of nodes based on the IsKindOf node type.

uint GetNodeOfTypeCount(UINT nodeType, bool fDisplayOnly = true)

{

uint cnt = 0;

for(CHierNode pNode = GetHead(fDisplayOnly); pNode; pNode = pNode.GetNext(fDisplayOnly))

if(pNode.IsKindOf(nodeType))

cnt++;

return cnt;

}

CHierNode GetFirstNodeOfType(UINT nodeType, bool fDisplayOnly = TRUE)

{

for(CHierNode pNode = GetHead(fDisplayOnly); pNode; pNode = pNode.GetNext(fDisplayOnly))

if(pNode.IsKindOf(nodeType))

return pNode;

return null;

}

CHierNode GetNextNodeOfType(UINT nodeType, CHierNode pPrevNode, bool fDisplayOnly = TRUE)

{

assert(pPrevNode && pPrevNode.IsKindOf(nodeType) && pPrevNode.GetParent() is this);

for(CHierNode pNode = pPrevNode.GetNext(fDisplayOnly); pNode; pNode = pNode.GetNext(fDisplayOnly))

if(pNode.IsKindOf(nodeType))

return pNode;

return null;

}

HRESULT GetConfigProvider(VARIANT \*pvar) { return E\_NOTIMPL; }

// Finds node

//CHierNode GetMatchingNode(LPCTSTR pszRelPath, bool fDisplayOnly = TRUE);

// Sorted list info

void SetIsSortedList(bool bValue) { SetBits(ST\_SortedList, bValue); }

bool IsSortedList() { return IsSet(ST\_SortedList); }

bool HaveChildrenBeenEnumerated() { return IsSet(ST\_ChildrenEnumerated); }

void SetChildrenBeenEnumerated(bool bValue) { SetBits(ST\_ChildrenEnumerated, bValue); }

protected:

void NotifyHierarchyOfAdd(CHierNode pNodeAdded)

{

if (HaveChildrenBeenEnumerated() && pNodeAdded.IsDisplayable())

{

CHierNode pNodePrev = GetCVsHierarchy().GetPrevDisplayableNode(pNodeAdded);

GetCVsHierarchy().OnItemAdded(this, pNodePrev, pNodeAdded);

}

}

static int CompareNodesForSort(CHierNode pNode1, CHierNode pNode2)

{

string name1 = (pNode1.IsContainer() ? "C" : "F") ~ pNode1.GetName();

string name2 = (pNode2.IsContainer() ? "C" : "F") ~ pNode2.GetName();

return CompareFilenames(name1, name2);

}

// Used by sorted lists.

void AddSorted(CHierNode pStartingNode, CHierNode pNode)

{

pNode.SetParent(this);

pNode.SetNext(null);

// Search for insertion point by doing an alpha comparison amongst the nodes

// If we are passed a start node, use it as the previous, and its ptr as the curNode,

// otherwise just start at the beginning.

CHierNode pCurNode = pStartingNode ? pStartingNode.GetNext(false) : m\_pHeadNode;

CHierNode pPrevNode = pStartingNode;

// Optimization to help project loads where the items are being added sorted, and there aren't really

// any duplicates. If no startingNode is specified, do a quick check against the tail to see if it belongs there

if(!pStartingNode && m\_pTailNode)

{

if (CompareNodesForSort(m\_pTailNode, pNode) < 0)

{

AddTail(pNode);

return;

}

}

while(pCurNode)

{ // ASSERT that there are not two items with the same name since the sorting relies on this.

if (CompareNodesForSort(pCurNode, pNode) > 0)

{ // Insert before this folder

if(pPrevNode)

{ // Inserting somewhere in the middle

pPrevNode.SetNext(pNode);

}

else

{ // Inserting at the head

m\_pHeadNode = pNode;

}

// Update who the just added node points to, we're done.

pNode.SetNext(pCurNode);

break;

}

pPrevNode = pCurNode;

pCurNode = pCurNode.GetNext(false);

}

// Past the end of the list, so this node becomes the new tail and maybe the new head too

if(!pCurNode)

{

if(!m\_pHeadNode)

{

assert(!m\_pTailNode && !pPrevNode);

m\_pHeadNode = pNode;

m\_pTailNode = pNode;

}

else

{

assert(m\_pTailNode && pPrevNode is m\_pTailNode);

m\_pTailNode.SetNext(pNode);

m\_pTailNode = pNode;

}

}

NotifyHierarchyOfAdd(pNode);

}

/////////////////////////////////

CHierNode m\_pHeadNode;

CHierNode m\_pTailNode;

}

// This file is part of Visual D

//

// Visual D integrates the D programming language into Visual Studio

// Copyright (c) 2010 by Rainer Schuetze, All Rights Reserved

//

// Distributed under the Boost Software License, Version 1.0.

// See accompanying file LICENSE\_1\_0.txt or copy at <http://www.boost.org/LICENSE_1_0.txt>

module visuald.build;

import visuald.comutil;

import visuald.logutil;

import visuald.chiernode;

import visuald.dproject;

import visuald.hierutil;

import visuald.hierarchy;

import visuald.fileutil;

import visuald.stringutil;

import visuald.config;

import visuald.dpackage;

import visuald.windows;

import visuald.pkgutil;

import stdext.path;

import stdext.file;

import stdext.string;

import stdext.array;

import core.stdc.stdlib;

import std.windows.charset;

import std.utf;

import std.string;

import std.file;

import std.path;

import std.conv;

import std.math;

import std.array;

import std.exception;

import std.algorithm;

import core.demangle;

import core.thread;

import core.stdc.time;

import core.stdc.string;

import std.regex;

//import stdext.fred;

import sdk.vsi.vsshell;

import sdk.vsi.vsshell80;

import sdk.vsi.vsshell90;

import xml = visuald.xmlwrap;

// threaded builds cause Visual Studio to close the solution

// version = threadedBuild;

// version = taskedBuild;

version(taskedBuild)

{

import std.parallelism;

}

// builder thread class

class CBuilderThread // : public CVsThread<CMyProjBuildableCfg>

{

public:

this(Config cfg)

{

mConfig = cfg;

// get a pointer to IVsLaunchPadFactory

m\_srpIVsLaunchPadFactory = queryService!(IVsLaunchPadFactory);

}

~this()

{

}

void Dispose()

{

m\_pIVsOutputWindowPane = release(m\_pIVsOutputWindowPane);

m\_srpIVsLaunchPadFactory = release(m\_srpIVsLaunchPadFactory);

m\_pIVsStatusbar = release(m\_pIVsStatusbar);

}

enum Operation

{

eIdle,

eBuild,

eRebuild,

eCheckUpToDate,

eClean,

};

HRESULT Start(Operation op, IVsOutputWindowPane pIVsOutputWindowPane)

{

logCall("%s.Start(op=%s, pIVsOutputWindowPane=%s)", this, op, cast(void\*\*) pIVsOutputWindowPane);

//mixin(LogCallMix2);

m\_op = op;

m\_pIVsOutputWindowPane = release(m\_pIVsOutputWindowPane);

m\_pIVsOutputWindowPane = addref(pIVsOutputWindowPane);

// Note that the QueryService for SID\_SVsStatusbar will fail during command line build

if(!m\_pIVsStatusbar)

m\_pIVsStatusbar = queryService!(IVsStatusbar);

mSuccess = true;

if(op == Operation.eCheckUpToDate)

ThreadMain(); // synchronous handling needed

else

{

version(taskedBuild)

{

auto task = task((CBuilderThread t) { t.ThreadMain(); }, this);

taskPool.put(task);

}

else version(threadedBuild)

{

mThread = new Thread(&ThreadMain);

mThread.start();

}

else

ThreadMain();

}

//return super::Start(pCMyProjBuildableCfg);

return mSuccess ? S\_OK : S\_FALSE;

}

void Stop(BOOL fSync)

{

mixin(LogCallMix2);

m\_fStopBuild = TRUE;

}

void QueryStatus(BOOL \*pfDone)

{

if(pfDone)

\*pfDone = (m\_op == Operation.eIdle);

}

void ThreadMain()

{

mixin(LogCallMix2);

BOOL fContinue = TRUE;

BOOL fSuccessfulBuild = FALSE; // set up for Fire\_BuildEnd() later on.

scope(exit)

{

version(threadedBuild)

mThread = null;

m\_op = Operation.eIdle;

}

m\_fStopBuild = false;

Fire\_BuildBegin(fContinue);

switch (m\_op)

{

default:

assert(\_false);

break;

case Operation.eBuild:

fSuccessfulBuild = DoBuild();

if(!fSuccessfulBuild)

StopSolutionBuild();

break;

case Operation.eRebuild:

fSuccessfulBuild = DoClean();

if(fSuccessfulBuild)

fSuccessfulBuild = DoBuild();

if(!fSuccessfulBuild)

StopSolutionBuild();

break;

case Operation.eCheckUpToDate:

fSuccessfulBuild = DoCheckIsUpToDate();

break;

case Operation.eClean:

fSuccessfulBuild = DoClean();

break;

}

Fire\_BuildEnd(fSuccessfulBuild);

mSuccess = fSuccessfulBuild != 0;

}

bool isStopped() const { return m\_fStopBuild != 0; }

bool needsOutputParser() { return true; }

string GetBuildDir()

{

return mConfig.GetProjectDir();

}

//////////////////////////////////////////////////////////////////////

static struct FileDep

{

CFileNode file;

string outfile;

string[] dependencies;

}

// sorts inplace

static void sortDependencies(FileDep[] filedeps)

{

for(int i = 0, j, k; i < filedeps.length; i++)

{

// sort i-th file before the first file that depends on it

for(j = 0; j < i; j++)

{

if(countUntil(filedeps[j].dependencies, filedeps[i].outfile) >= 0)

break;

}

// check whether the i-th file depends on any later file

for(k = j; k < i; k++)

{

if(countUntil(filedeps[i].dependencies, filedeps[k].outfile) >= 0)

throw new Exception("circular dependency on " ~ filedeps[i].outfile);

}

if(j < i)

{

FileDep dep = filedeps[i];

for(k = i; k > j; k--)

filedeps[k] = filedeps[k-1];

filedeps[j] = dep;

}

}

}

CFileNode[] BuildDependencyList()

{

string workdir = mConfig.GetProjectDir();

Config config = mConfig; // closure does not work with both local variables and this pointer?

FileDep[] filedeps;

CHierNode node = searchNode(mConfig.GetProjectNode(),

delegate (CHierNode n) {

if(CFileNode file = cast(CFileNode) n)

{

string tool = config.GetCompileTool(file);

if(tool == "Custom" || tool == kToolResourceCompiler || tool == kToolCpp)

{

FileDep dep;

dep.outfile = config.GetOutputFile(file);

dep.outfile = canonicalPath(makeFilenameAbsolute(dep.outfile, workdir));

dep.dependencies = config.GetDependencies(file);

foreach(ref d; dep.dependencies)

d = canonicalPath(d);

dep.file = file;

filedeps ~= dep;

}

}

return false;

});

sortDependencies(filedeps);

CFileNode[] files;

foreach(fdep; filedeps)

files ~= fdep.file;

return files;

}

unittest

{

FileDep[] deps = [

{ null, "file1", [ "file2", "file3" ] },

{ null, "file2", [ "file4", "file5" ] },

{ null, "file3", [ "file2", "file6" ] },

];

sortDependencies(deps);

assert(deps[0].outfile == "file2");

assert(deps[1].outfile == "file3");

assert(deps[2].outfile == "file1");

deps[0].dependencies ~= "file1";

try

{

sortDependencies(deps);

assert(false);

}

catch(Exception e)

{

assert(std.string.indexOf(e.msg, "circular") >= 0);

}

}

//////////////////////////////////////////////////////////////////////

bool buildCustomFile(CFileNode file, ref bool built)

{

string reason;

if(!mConfig.isUptodate(file, &reason))

{

string cmdline = mConfig.GetCompileCommand(file);

if(cmdline.length)

{

string workdir = mConfig.GetProjectDir();

string outfile = mConfig.GetOutputFile(file);

string cmdfile = makeFilenameAbsolute(outfile ~ "." ~ kCmdLogFileExtension, workdir);

showUptodateFailure(reason, outfile);

removeCachedFileTime(makeFilenameAbsolute(outfile, workdir));

HRESULT hr = RunCustomBuildBatchFile(outfile, cmdfile, cmdline, m\_pIVsOutputWindowPane, this);

if (hr != S\_OK)

return false; // stop compiling

}

built = true;

}

return true;

}

//////////////////////////////////////////////////////////////////////

bool buildPhobos(ref bool built)

{

string reason;

if(!mConfig.isPhobosUptodate(&reason))

{

string cmdline = mConfig.GetPhobosCommandLine();

if(cmdline.length)

{

string workdir = mConfig.GetProjectDir();

string outfile = mConfig.GetPhobosPath();

string cmdfile = makeFilenameAbsolute(outfile ~ "." ~ kCmdLogFileExtension, workdir);

showUptodateFailure(reason, outfile);

removeCachedFileTime(makeFilenameAbsolute(outfile, workdir));

HRESULT hr = RunCustomBuildBatchFile(outfile, cmdfile, cmdline, m\_pIVsOutputWindowPane, this);

if (hr != S\_OK)

return false; // stop compiling

}

built = true;

}

return true;

}

/\*\* build non-D files \*/

bool doCustomBuilds(out bool hasCustomBuilds, out int numCustomBuilds)

{

mixin(LogCallMix2);

bool built;

if(mConfig.GetProjectOptions().privatePhobos)

{

if (!buildPhobos(built))

return false;

if(built)

numCustomBuilds++;

}

// first build custom files with dependency graph

CFileNode[] files = BuildDependencyList();

foreach(file; files)

{

if(isStopped())

return false;

if(!buildCustomFile(file, built))

return false;

hasCustomBuilds = true;

if(built)

numCustomBuilds++;

}

// now build files not in the dependency graph (d files in single compilation modes)

CHierNode node = searchNode(mConfig.GetProjectNode(),

delegate (CHierNode n) {

if(CFileNode file = cast(CFileNode) n)

{

if(files.contains(file))

return false;

if(isStopped())

return true;

if(!buildCustomFile(file, built))

return true;

hasCustomBuilds = true;

if(built)

numCustomBuilds++;

}

return false;

});

return node is null;

}

bool DoBuild()

{

mixin(LogCallMix2);

beginLog();

HRESULT hr = S\_FALSE;

try

{

string target = mConfig.GetTargetPath();

string msg = "Building " ~ target ~ "...\n";

if(m\_pIVsOutputWindowPane)

{

ScopedBSTR bstrMsg = ScopedBSTR(msg);

m\_pIVsOutputWindowPane.OutputString(bstrMsg);

}

string workdir = mConfig.GetProjectDir();

string outdir = makeFilenameAbsolute(mConfig.GetOutDir(), workdir);

if(!exists(outdir))

mkdirRecurse(outdir);

string intermediatedir = makeFilenameAbsolute(mConfig.GetIntermediateDir(), workdir);

if(!exists(intermediatedir))

mkdirRecurse(intermediatedir);

string modules\_ddoc;

if(mConfig.getModulesDDocCommandLine([], modules\_ddoc).length)

{

modules\_ddoc = unquoteArgument(modules\_ddoc);

modules\_ddoc = mConfig.GetProjectOptions().replaceEnvironment(modules\_ddoc, mConfig);

string modpath = dirName(modules\_ddoc);

modpath = makeFilenameAbsolute(modpath, workdir);

if(!exists(modpath))

mkdirRecurse(modpath);

}

bool hasCustomBuilds;

int numCustomBuilds;

if(!doCustomBuilds(hasCustomBuilds, numCustomBuilds))

return false;

if(hasCustomBuilds)

if(targetIsUpToDate()) // only recheck target if custom builds exist

return true; // no need to rebuild target if custom builds did not change target dependencies

if(!mLastUptodateFailure.empty)

showUptodateFailure(mLastUptodateFailure);

string cmdline = mConfig.getCommandLine();

string cmdfile = makeFilenameAbsolute(mConfig.GetCommandLinePath(), workdir);

hr = RunCustomBuildBatchFile(target, cmdfile, cmdline, m\_pIVsOutputWindowPane, this);

if(hr == S\_OK && mConfig.GetProjectOptions().compilationModel == ProjectOptions.kSingleFileCompilation)

mConfig.writeLinkDependencyFile();

return (hr == S\_OK);

}

catch(Exception e)

{

OutputText("Error setting up build: " ~ e.msg);

return false;

}

finally

{

endLog(hr == S\_OK);

}

}

bool customFilesUpToDate()

{

if(mConfig.GetProjectOptions().privatePhobos)

if (!mConfig.isPhobosUptodate(null))

return false;

CHierNode node = searchNode(mConfig.GetProjectNode(),

delegate (CHierNode n)

{

if(isStopped())

return true;

if(CFileNode file = cast(CFileNode) n)

{

if(!mConfig.isUptodate(file, null))

return true;

}

return false;

});

return node is null;

}

bool getTargetDependencies(ref string[] files)

{

string workdir = mConfig.GetProjectDir();

string deppath = makeFilenameAbsolute(mConfig.GetDependenciesPath(), workdir);

if(!std.file.exists(deppath))

return showUptodateFailure("dependency file " ~ deppath ~ " does not exist");

if(!getFilenamesFromDepFile(deppath, files))

return showUptodateFailure("dependency file " ~ deppath ~ " cannot be read");

if(mConfig.hasLinkDependencies())

{

string lnkdeppath = makeFilenameAbsolute(mConfig.GetLinkDependenciesPath(), workdir);

if(!std.file.exists(lnkdeppath))

return showUptodateFailure("link dependency file " ~ lnkdeppath ~ " does not exist");

string lnkdeps;

auto lnkdepData = cast(ubyte[])std.file.read(lnkdeppath);

if(lnkdepData.length > 1 && lnkdepData[0] == 0xFF && lnkdepData[1] == 0xFE)

{

wstring lnkdepw = cast(wstring)lnkdepData[2..$];

lnkdeps = to!string(lnkdepw);

}

else

{

auto lnkdepz = cast(string)lnkdepData ~ "\0";

int cp = GetKBCodePage();

lnkdeps = fromMBSz(lnkdepz.ptr, cp);

}

string[] lnkfiles = splitLines(lnkdeps);

foreach(lnkfile; lnkfiles)

{

if(!lnkfile.startsWith("#Command:"))

files ~= makeFilenameAbsolute(lnkfile, workdir);

}

}

return true;

}

bool DoCheckIsUpToDate()

{

mixin(LogCallMix2);

clearCachedFileTimes();

scope(exit) clearCachedFileTimes();

mLastUptodateFailure = null;

if(!customFilesUpToDate())

return false;

return targetIsUpToDate();

}

bool targetIsUpToDate()

{

string workdir = mConfig.GetProjectDir();

string cmdfile = makeFilenameAbsolute(mConfig.GetCommandLinePath(), workdir);

string cmdline = mConfig.getCommandLine();

if(!compareCommandFile(cmdfile, cmdline))

return showUptodateFailure("command line changed");

string target = makeFilenameAbsolute(mConfig.GetTargetPath(), workdir);

string oldestFile;

long targettm = getOldestFileTime( [ target ], oldestFile );

if(targettm == long.min)

return showUptodateFailure("target does not exist");

string[] files;

if(!getTargetDependencies(files))

return false;

string[] libs = mConfig.getLibsFromDependentProjects();

files ~= libs;

makeFilenamesAbsolute(files, workdir);

string newestFile;

long sourcetm = getNewestFileTime(files, newestFile);

if(targettm <= sourcetm)

return showUptodateFailure("older than " ~ newestFile);

return true;

}

bool DoClean()

{

mixin(LogCallMix2);

string[] files = mConfig.GetBuildFiles();

foreach(string file; files)

{

try

{

if(std.string.indexOf(file,'\*') >= 0 || std.string.indexOf(file,'?') >= 0)

{

string dir = dirName(file);

string pattern = baseName(file);

if(isExistingDir(dir))

foreach(string f; dirEntries(dir, SpanMode.depth))

if(globMatch(f, pattern))

std.file.remove(f);

}

else if(std.file.exists(file))

std.file.remove(file);

}

catch(FileException e)

{

OutputText("cannot delete " ~ file ~ ":" ~ e.msg);

}

}

return true;

}

void OutputText(string msg)

{

wchar\* wmsg = \_toUTF16z(msg);

if (m\_pIVsStatusbar)

{

m\_pIVsStatusbar.SetText(wmsg);

}

if (m\_pIVsOutputWindowPane)

{

m\_pIVsOutputWindowPane.OutputString(wmsg);

m\_pIVsOutputWindowPane.OutputString(cast(wchar\*)"\n"w.ptr);

}

}

/+

void InternalTick(ref BOOL rfContine);

+/

void Fire\_Tick(ref BOOL rfContinue)

{

rfContinue = mConfig.FFireTick() && !m\_fStopBuild;

}

void Fire\_BuildBegin(ref BOOL rfContinue)

{

mixin(LogCallMix2);

mConfig.FFireBuildBegin(rfContinue);

}

void Fire\_BuildEnd(BOOL fSuccess)

{

mixin(LogCallMix2);

mConfig.FFireBuildEnd(fSuccess);

}

void StopSolutionBuild()

{

if(!Package.GetGlobalOptions().stopSolutionBuild)

return;

if(auto solutionBuildManager = queryService!(IVsSolutionBuildManager)())

{

OutputText("Solution build stopped.");

scope(exit) release(solutionBuildManager);

solutionBuildManager.CancelUpdateSolutionConfiguration();

}

}

bool showUptodateFailure(string msg, string target = null)

{

if(!m\_pIVsOutputWindowPane)

mLastUptodateFailure = msg;

else if(Package.GetGlobalOptions().showUptodateFailure)

{

if(target.empty)

target = mConfig.GetTargetPath();

msg = target ~ " not up to date: " ~ msg;

OutputText(msg); // writeToBuildOutputPane

}

return false;

}

void beginLog()

{

mStartBuildTime = time(null);

mBuildLog = `<html><head><META HTTP-EQUIV="Content-Type" content="text/html">

</head><body><pre>

<table width=100% bgcolor=#CFCFE5><tr><td>

<font face=arial size=+3>Build Log</font>

</table>

`;

}

void addCommandLog(string target, string cmd, string output)

{

if(!mCreateLog)

return;

mBuildLog ~= "<table width=100% bgcolor=#DFDFE5><tr><td><font face=arial size=+2>\n";

mBuildLog ~= xml.encode("Building " ~ target);

mBuildLog ~= "\n</font></table>\n";

mBuildLog ~= "<table width=100% bgcolor=#EFEFE5><tr><td><font face=arial size=+1>\n";

mBuildLog ~= "Command Line";

mBuildLog ~= "\n</font></table>\n";

mBuildLog ~= xml.encode(cmd);

mBuildLog ~= "<table width=100% bgcolor=#EFEFE5><tr><td><font face=arial size=+1>\n";

mBuildLog ~= "Output";

mBuildLog ~= "\n</font></table>\n";

mBuildLog ~= xml.encode(output) ~ "\n";

}

void endLog(bool success)

{

if(!mCreateLog)

return;

mBuildLog ~= "</body></html>";

string workdir = mConfig.GetProjectDir();

string intdir = makeFilenameAbsolute(mConfig.GetIntermediateDir(), workdir);

string logfile = mConfig.GetBuildLogFile();

try

{

std.file.write(logfile, mBuildLog);

if(!success)

OutputText("Details saved as \"[file://](NULL)" ~ logfile ~ "\"");

}

catch(FileException e)

{

OutputText("cannot write " ~ logfile ~ ":" ~ e.msg);

}

if(Package.GetGlobalOptions().timeBuilds)

{

time\_t now = time(null);

double duration = difftime(now, mStartBuildTime);

if(duration >= 60)

{

int min = cast(int) floor(duration / 60);

int sec = cast(int) floor(duration - 60 \* min);

string tm = format("%d:%02d", min, sec);

OutputText("Build time: " ~ to!string(min) ~ ":" ~ to!string(sec) ~ " min");

}

else

OutputText("Build time: " ~ to!string(duration) ~ " s");

}

}

/+

virtual HRESULT PrepareInStartingThread(CMyProjBuildableCfg \*pCMyProjBuildableCfg);

virtual HRESULT InnerThreadMain(CMyProjBuildableCfg \*pBuildableCfg);

virtual void ReleaseThreadHandle();

+/

Config mConfig;

IVsLaunchPadFactory m\_srpIVsLaunchPadFactory;

IStream m\_pIStream\_IVsOutputWindowPane;

IVsOutputWindowPane m\_pIVsOutputWindowPane;

IStream m\_pIStream\_IVsStatusbar;

IVsStatusbar m\_pIVsStatusbar;

BOOL m\_fIsUpToDate;

Operation m\_op;

BOOL m\_fStopBuild;

HANDLE m\_hEventStartSync;

time\_t mStartBuildTime;

version(threadedBuild)

Thread mThread; // keep a reference to the thread to avoid it from being collected

bool mSuccess = false;

bool mCreateLog = true;

string mBuildLog;

string mLastUptodateFailure;

};

class CLaunchPadEvents : DComObject, IVsLaunchPadEvents

{

this(CBuilderThread builder)

{

m\_pBuilder = builder;

}

override HRESULT QueryInterface(in IID\* riid, void\*\* pvObject)

{

if(queryInterface!(IVsLaunchPadEvents) (this, riid, pvObject))

return S\_OK;

return super.QueryInterface(riid, pvObject);

}

// IVsLaunchPadEvents

override HRESULT Tick(/\* [out] \*/ BOOL \* pfCancel)

{

BOOL fContinue = TRUE;

m\_pBuilder.Fire\_Tick(fContinue);

\*pfCancel = !fContinue;

return S\_OK;

}

public:

CBuilderThread m\_pBuilder;

};

string demangleText(string ln)

{

string txt;

static if(\_\_traits(compiles, (){uint p; decodeDmdString("", p);}))

uint i;

else

int i; // until dmd 2.056

for (i = 0; i < ln.length; )

{

char ch = ln[i]; // compressed symbols are NOT utf8!

if(isAlphaNum(ch) || ch == '\_')

{

string s = decodeDmdString(ln, i);

if(s.length > 3 && s[0] == '\_' && s[1] == 'D' && isDigit(s[2]))

{

auto d = core.demangle.demangle(s);

txt ~= d;

}

else if(s.length > 4 && s[0] == '\_' && s[1] == '\_' && s[2] == 'D' && isDigit(s[3]))

{

// \_\_moddtor/\_\_modctor have duplicate '\_\_'

auto d = core.demangle.demangle(s[1..$]);

if(d == s[1..$])

txt ~= s;

else

txt ~= d;

}

else

txt ~= s;

}

else

{

txt ~= ch;

i++;

}

}

return txt;

}

class CLaunchPadOutputParser : DComObject, IVsLaunchPadOutputParser

{

this(CBuilderThread builder)

{

mCompiler = builder.mConfig.GetProjectOptions().compiler;

mProjectDir = builder.mConfig.GetProjectDir();

}

override HRESULT QueryInterface(in IID\* riid, void\*\* pvObject)

{

if(queryInterface!(IVsLaunchPadOutputParser) (this, riid, pvObject))

return S\_OK;

return super.QueryInterface(riid, pvObject);

}

override HRESULT ParseOutputStringForInfo(

in LPCOLESTR pszOutputString, // one line of output text

/+[out, optional]+/ BSTR \*pbstrFilename, // fully-qualified file name for task list item (may be NULL)

/+[out, optional]+/ ULONG \*pnLineNum, // file line number for task list item (may be NULL)

/+[out, optional]+/ ULONG \*pnPriority, // priority for task list item (may be NULL)

/+[out, optional]+/ BSTR \*pbstrTaskItemText, // description text for task list item (may be NULL)

/+[out, optional]+/ BSTR \*pbstrHelpKeyword)

{

mixin(LogCallMix2);

string line = to\_string(pszOutputString);

uint nPriority, nLineNum;

string filename, taskItemText;

if(!parseOutputStringForTaskItem(line, nPriority, filename, nLineNum, taskItemText, mCompiler))

return S\_FALSE;

//if(Package.GetGlobalOptions().demangleError)

//        taskItemText = demangleText(taskItemText);

filename = makeFilenameCanonical(filename, mProjectDir);

if(pnPriority)

\*pnPriority = nPriority;

if(pnLineNum)

\*pnLineNum = nLineNum - 1;

if(pbstrFilename)

\*pbstrFilename = allocBSTR(filename);

if(pbstrTaskItemText)

\*pbstrTaskItemText = allocBSTR(taskItemText);

return S\_OK;

}

string mProjectDir;

int mCompiler;

}

// Runs the build commands, writing cmdfile if successful

HRESULT RunCustomBuildBatchFile(string target,

string buildfile,

string cmdline,

IVsOutputWindowPane pIVsOutputWindowPane,

CBuilderThread pBuilder)

{

logCall("RunCustomBuildBatchFile(target=\"%s\", buildfile=\"%s\")", target, buildfile);

if (cmdline.length == 0)

return S\_OK;

HRESULT hr = S\_OK;

// get the project root directory.

string strBuildDir = pBuilder.GetBuildDir();

string batchFileText = insertCr(cmdline);

string output;

Package.GetGlobalOptions().addBuildPath(strBuildDir);

string cmdfile = buildfile ~ ".cmd";

assert(pBuilder.m\_srpIVsLaunchPadFactory);

ComPtr!(IVsLaunchPad) srpIVsLaunchPad;

hr = pBuilder.m\_srpIVsLaunchPadFactory.CreateLaunchPad(&srpIVsLaunchPad.ptr);

scope(exit) pBuilder.addCommandLog(target, cmdline, output);

if(FAILED(hr))

{

output = format("internal error: IVsLaunchPadFactory.CreateLaunchPad failed with rc=%x", hr);

return hr;

}

assert(srpIVsLaunchPad.ptr);

CLaunchPadEvents pLaunchPadEvents = newCom!CLaunchPadEvents(pBuilder);

BSTR bstrOutput;

version(none)

{

hr = srpIVsLaunchPad.ExecBatchScript(

/\* [in] LPCOLESTR pszBatchFileContents \*/ \_toUTF16z(batchFileText),

/\* [in] LPCOLESTR pszWorkingDir \*/ \_toUTF16z(strBuildDir), // may be NULL, passed on to CreateProcess (wee Win32 API for details)

/\* [in] LAUNCHPAD\_FLAGS lpf \*/ LPF\_PipeStdoutToOutputWindow,

/\* [in] IVsOutputWindowPane \*pOutputWindowPane \*/ pIVsOutputWindowPane, // if LPF\_PipeStdoutToOutputWindow, which pane in the output window should the output be piped to

/\* [in] ULONG nTaskItemCategory \*/ 0, // if LPF\_PipeStdoutToTaskList is specified

/\* [in] ULONG nTaskItemBitmap \*/ 0, // if LPF\_PipeStdoutToTaskList is specified

/\* [in] LPCOLESTR pszTaskListSubcategory \*/ null, // if LPF\_PipeStdoutToTaskList is specified

/\* [in] IVsLaunchPadEvents \*pVsLaunchPadEvents \*/ pLaunchPadEvents,

/\* [out] BSTR \*pbstrOutput \*/ &bstrOutput); // all output generated (may be NULL)

if(FAILED(hr))

{

output = format("internal error: IVsLaunchPad.ptr.ExecBatchScript failed with rc=%x", hr);

return hr;

}

} else {

try

{

int cp = GetKBCodePage();

const(char)\*p = toMBSz(batchFileText, cp);

int plen = strlen(p);

string dir = dirName(cmdfile);

if(!std.file.exists(dir))

mkdirRecurse(dir);

std.file.write(cmdfile, p[0..plen]);

}

catch(FileException e)

{

output = format("internal error: cannot write file " ~ cmdfile);

hr = S\_FALSE;

}

DWORD result;

IVsLaunchPad2 pad2 = qi\_cast!IVsLaunchPad2(srpIVsLaunchPad);

if(pad2 && pBuilder.needsOutputParser())

{

CLaunchPadOutputParser pLaunchPadOutputParser = newCom!CLaunchPadOutputParser(pBuilder);

hr = pad2.ExecCommandEx(

/\* [in] LPCOLESTR pszApplicationName \*/ \_toUTF16z(getCmdPath()),

/\* [in] LPCOLESTR pszCommandLine \*/ \_toUTF16z("/Q /C " ~ quoteFilename(cmdfile)),

/\* [in] LPCOLESTR pszWorkingDir \*/ \_toUTF16z(strBuildDir), // may be NULL, passed on to CreateProcess (wee Win32 API for details)

/\* [in] LAUNCHPAD\_FLAGS lpf \*/ LPF\_PipeStdoutToOutputWindow | LPF\_PipeStdoutToTaskList,

/\* [in] IVsOutputWindowPane \*pOutputWindowPane \*/ pIVsOutputWindowPane, // if LPF\_PipeStdoutToOutputWindow, which pane in the output window should the output be piped to

/\* [in] ULONG nTaskItemCategory \*/ CAT\_BUILDCOMPILE, // if LPF\_PipeStdoutToTaskList is specified

/\* [in] ULONG nTaskItemBitmap \*/ 0, // if LPF\_PipeStdoutToTaskList is specified

/\* [in] LPCOLESTR pszTaskListSubcategory \*/ null, // "Build"w.ptr, // if LPF\_PipeStdoutToTaskList is specified

/\* [in] IVsLaunchPadEvents pVsLaunchPadEvents \*/ pLaunchPadEvents,

/\* [in] IVsLaunchPadOutputParser pOutputParser \*/ pLaunchPadOutputParser,

/\* [out] DWORD \*pdwProcessExitCode \*/ &result,

/\* [out] BSTR \*pbstrOutput \*/ &bstrOutput); // all output generated (may be NULL)

release(pad2);

}

else

hr = srpIVsLaunchPad.ExecCommand(

/\* [in] LPCOLESTR pszApplicationName \*/ \_toUTF16z(getCmdPath()),

/\* [in] LPCOLESTR pszCommandLine \*/ \_toUTF16z("/Q /C " ~ quoteFilename(cmdfile)),

/\* [in] LPCOLESTR pszWorkingDir \*/ \_toUTF16z(strBuildDir), // may be NULL, passed on to CreateProcess (wee Win32 API for details)

/\* [in] LAUNCHPAD\_FLAGS lpf \*/ LPF\_PipeStdoutToOutputWindow | LPF\_PipeStdoutToTaskList,

/\* [in] IVsOutputWindowPane \*pOutputWindowPane \*/ pIVsOutputWindowPane, // if LPF\_PipeStdoutToOutputWindow, which pane in the output window should the output be piped to

/\* [in] ULONG nTaskItemCategory \*/ CAT\_BUILDCOMPILE, // if LPF\_PipeStdoutToTaskList is specified

/\* [in] ULONG nTaskItemBitmap \*/ 0, // if LPF\_PipeStdoutToTaskList is specified

/\* [in] LPCOLESTR pszTaskListSubcategory \*/ null, // "Build"w.ptr, // if LPF\_PipeStdoutToTaskList is specified

/\* [in] IVsLaunchPadEvents \*pVsLaunchPadEvents \*/ pLaunchPadEvents,

/\* [out] DWORD \*pdwProcessExitCode \*/ &result,

/\* [out] BSTR \*pbstrOutput \*/ &bstrOutput); // all output generated (may be NULL)

if(FAILED(hr))

{

output = format("internal error: IVsLaunchPad.ptr.ExecCommand failed with rc=%x", hr);

return hr;

}

if(result != 0)

hr = S\_FALSE;

}

// don't know how to get at the exit code, so check output string

output = strip(detachBSTR(bstrOutput));

if(hr == S\_OK && \_endsWith(output, "failed!"))

hr = S\_FALSE;

// outputToErrorList(srpIVsLaunchPad, pBuilder, pIVsOutputWindowPane, output);

if(hr == S\_OK)

{

try

{

std.file.write(buildfile, cmdline);

}

catch(FileException e)

{

output = format("internal error: cannot write file " ~ buildfile);

hr = S\_FALSE;

}

}

return hr;

}

HRESULT outputToErrorList(IVsLaunchPad pad, CBuilderThread pBuilder,

IVsOutputWindowPane outPane, string output)

{

logCall("outputToErrorList(output=\"%s\")", output);

HRESULT hr;

auto prj = \_toUTF16z(pBuilder.mConfig.GetProjectPath());

string[] lines = std.string.split(output, "\n");

foreach(line; lines)

{

uint nPriority, nLineNum;

string strFilename, strTaskItemText;

if(parseOutputStringForTaskItem(line, nPriority, strFilename, nLineNum, strTaskItemText, Compiler.DMD))

{

IVsOutputWindowPane2 pane2 = qi\_cast!IVsOutputWindowPane2(outPane);

if(pane2)

hr = pane2.OutputTaskItemStringEx2(

"."w.ptr, // The text to write to the output window.

nPriority, // The priority: use TP\_HIGH for errors.

CAT\_BUILDCOMPILE, // Not used internally; pass NULL unless you want to use it for your own purposes.

null, // Not used internally; pass NULL unless you want to use it for your own purposes.

0, // Not used internally.

\_toUTF16z(strFilename), // The file name for the Error List entry; may be NULL if no file is associated with the error.

nLineNum, // Zero-based line number in pszFilename.

nLineNum, // Zero-based column in pszFilename.

prj, // The unique name of the project for the Error List entry; may be NULL if no project is associated with the error.

\_toUTF16z(strTaskItemText), // The text of the Error List entry.

""w.ptr); // in LPCOLESTR pszLookupKwd

else // no project or column +/

hr = outPane.OutputTaskItemStringEx(

" "w.ptr, // The text to write to the output window.

nPriority, // The priority: use TP\_HIGH for errors.

CAT\_BUILDCOMPILE, // Not used internally; pass NULL unless you want to use it for your own purposes.

null, // Not used internally; pass NULL unless you want to use it for your own purposes.

0, // Not used internally.

\_toUTF16z(strFilename), // The file name for the Error List entry; may be NULL if no file is associated with the error.

nLineNum, // Zero-based line number in pszFilename.

\_toUTF16z(strTaskItemText), // The text of the Error List entry.

""w.ptr); // in LPCOLESTR pszLookupKwd

release(pane2);

}

}

return hr;

}

bool isInitializedRE(T)(ref T re)

{

static if(\_\_traits(compiles,re.ir))

return re.ir !is null; // stdext.fred

else

return re.captures() > 0; // std.regex

}

bool parseOutputStringForTaskItem(string outputLine, out uint nPriority,

out string filename, out uint nLineNum,

out string itemText, int compiler)

{

outputLine = strip(outputLine);

// DMD compile error

\_\_gshared static Regex!char re1dmd, re1gdc, remixin, re2, re3, re4, re5, re6;

if(!isInitializedRE(remixin))

remixin = regex(r"^(.\*?)-mixin-([0-9]+)\(([0-9]+)\):(.\*)$");

auto rematch = match(outputLine, remixin);

if(!rematch.empty())

{

auto captures = rematch.captures();

filename = replace(captures[1], "\\\\", "\\");

string lineno = captures[2];

string lineno2 = captures[3];

nLineNum = to!uint(lineno);

uint nMixinLine = to!uint(lineno2) - nLineNum + 1;

itemText = "mixin(" ~ to!string(nMixinLine) ~ ") " ~ strip(captures[4]);

if(itemText.startsWith("Warning:")) // make these errors if not building with -wi?

nPriority = TP\_NORMAL;

else

nPriority = TP\_HIGH;

return true;

}

// exception/error when running

if(!isInitializedRE(re5))

re5 = regex(r"^[^ @]\*@(.\*?)\(([0-9]+)\):(.\*)$");

rematch = match(outputLine, re5);

if(!rematch.empty())

{

auto captures = rematch.captures();

nPriority = TP\_HIGH;

filename = replace(captures[1], "\\\\", "\\");

string lineno = captures[2];

nLineNum = to!uint(lineno);

itemText = strip(captures[3]);

return true;

}

if(!isInitializedRE(re1dmd))

re1dmd = regex(r"^(.\*?)\(([0-9]+)\):(.\*)$"); // replace . with [\x00-\x7f] for std.regex

if(!isInitializedRE(re1gdc))

re1gdc = regex(r"^(.\*?):([0-9]+):(.\*)$");

rematch = match(outputLine, compiler == Compiler.GDC ? re1gdc : re1dmd);

if(!rematch.empty())

{

auto captures = rematch.captures();

filename = replace(captures[1], "\\\\", "\\");

string lineno = captures[2];

nLineNum = to!uint(lineno);

itemText = strip(captures[3]);

if(itemText.startsWith("Warning:")) // make these errors if not building with -wi?

nPriority = TP\_NORMAL;

else

nPriority = TP\_HIGH;

return true;

}

// link error

if(!isInitializedRE(re2))

re2 = regex(r"^ \*(Error \*[0-9]+:.\*)$");

rematch = match(outputLine, re2);

if(!rematch.empty())

{

nPriority = TP\_HIGH;

filename = "";

nLineNum = 0;

itemText = strip(rematch.captures[1]);

return true;

}

// link error with file name

if(!isInitializedRE(re3))

re3 = regex(r"^(.\*?)\(([0-9]+)\) \*: \*(Error \*[0-9]+:.\*)$");

rematch = match(outputLine, re3);

if(!rematch.empty())

{

auto captures = rematch.captures();

nPriority = TP\_HIGH;

filename = replace(captures[1], "\\\\", "\\");

string lineno = captures[2];

nLineNum = to!uint(lineno);

itemText = strip(captures[3]);

return true;

}

// link warning

if(!isInitializedRE(re4))

re4 = regex(r"^ \*(Warning \*[0-9]+:.\*)$");

rematch = match(outputLine, re4);

if(!rematch.empty())

{

nPriority = TP\_NORMAL;

filename = "";

nLineNum = 0;

itemText = strip(rematch.captures[1]);

return true;

}

// entry in exception call stack

if(!isInitializedRE(re6))

re6 = regex(r"^0x[0-9a-fA-F]\* in .\* at (.\*?)\(([0-9]+)\)(.\*)$");

rematch = match(outputLine, re6);

if(!rematch.empty())

{

auto captures = rematch.captures();

nPriority = TP\_LOW;

filename = replace(captures[1], "\\\\", "\\");

string lineno = captures[2];

nLineNum = to!uint(lineno);

itemText = strip(captures[3]);

return true;

}

return false;

}

unittest

{

uint nPriority, nLineNum;

string strFilename, strTaskItemText;

bool rc = parseOutputStringForTaskItem("file.d(37): huhu", nPriority, strFilename, nLineNum, strTaskItemText, Compiler.DMD);

assert(rc);

assert(strFilename == "file.d");

assert(nLineNum == 37);

assert(strTaskItemText == "huhu");

rc = parseOutputStringForTaskItem("main.d(10): Error: undefined identifier A, did you mean B?",

nPriority, strFilename, nLineNum, strTaskItemText, Compiler.DMD);

assert(rc);

assert(strFilename == "main.d");

assert(nLineNum == 10);

assert(strTaskItemText == "Error: undefined identifier A, did you mean B?");

rc = parseOutputStringForTaskItem(r"object.Exception@C:\tmp\d\forever.d(28): what?",

nPriority, strFilename, nLineNum, strTaskItemText, Compiler.DMD);

assert(rc);

assert(strFilename == r"C:\tmp\d\forever.d");

assert(nLineNum == 28);

assert(strTaskItemText == "what?");

rc = parseOutputStringForTaskItem(r"0x004020C8 in void test.\_\_modtest() at C:\tmp\d\forever.d(34)",

nPriority, strFilename, nLineNum, strTaskItemText, Compiler.DMD);

assert(rc);

assert(strFilename == r"C:\tmp\d\forever.d");

assert(nLineNum == 34);

assert(strTaskItemText == "");

rc = parseOutputStringForTaskItem(r"D:\LuaD\luad\conversions\structs.d-mixin-36(36): Error: cast(MFVector)(\*\_this).x is not an lvalue",

nPriority, strFilename, nLineNum, strTaskItemText, Compiler.DMD);

assert(rc);

assert(strFilename == r"D:\LuaD\luad\conversions\structs.d");

assert(nLineNum == 36);

assert(strTaskItemText == "mixin(1) Error: cast(MFVector)(\*\_this).x is not an lvalue");

}

string unEscapeFilename(string file)

{

int pos = std.string.indexOf(file, '\\');

if(pos < 0)

return file;

char[] p;

int start = 0;

while(pos < file.length)

{

if(file[pos+1] == '(' || file[pos+1] == ')' || file[pos+1] == '\\')

{

p ~= file[start .. pos];

start = pos + 1;

}

int nextpos = std.string.indexOf(file[pos + 1 .. $], '\\');

if(nextpos < 0)

break;

pos += nextpos + 1;

}

p ~= file[start..$];

return assumeUnique(p);

}

string re\_match\_dep = r"^[A-Za-z0-9\_\.]+ \*\((.\*)\) : p[a-z]\* : [A-Za-z0-9\_\.]+ \((.\*)\)$";

bool getFilenamesFromDepFile(string depfile, ref string[] files)

{

// converted int[string] to byte[string] due to bug #2500

byte[string] aafiles;

int cntValid = 0;

try

{

string txt = cast(string)std.file.read(depfile);

version(slow)

{

RegExp re = new RegExp(re\_match\_dep);

string[] lines = splitLines(txt);

foreach(line; lines)

{

string[] match = re.exec(line);

if(match.length == 3)

{

string file1 = replace(match[1], "\\\\", "\\");

string file2 = replace(match[2], "\\\\", "\\");

aafiles[file1] = 1;

aafiles[file2] = 1;

cntValid++;

}

}

}

else

{

uint pos = 0;

uint openpos = 0;

bool skipNext = false;

bool stringImport = false;

while(pos < txt.length)

{

dchar ch = decode(txt, pos);

if(skipNext)

{

skipNext = false;

continue;

}

if(ch == '\\')

skipNext = true;

if(ch == '(')

openpos = pos;

else if(ch == ')' && openpos > 0)

{

// only check lines that import "object", these are written once per file

const string kCheck1 = " : public : object ";

const string kCheck2 = " : private : object "; // added after 2.060

const string kCheck3 = " : string : "; // string imports added after 2.064

if((pos + kCheck1.length <= txt.length && txt[pos .. pos + kCheck1.length] == kCheck1) ||

(pos + kCheck2.length <= txt.length && txt[pos .. pos + kCheck2.length] == kCheck2) ||

stringImport)

{

string file = txt[openpos .. pos-1];

file = unEscapeFilename(file);

aafiles[file] = 1;

openpos = 0;

stringImport = false;

cntValid++;

}

else if(pos + kCheck3.length <= txt.length && txt[pos .. pos + kCheck3.length] == kCheck3)

{

// wait for the next file name in () on the same line

openpos = 0;

stringImport = true;

}

}

else if(ch == '\n')

{

openpos = 0;

stringImport = false;

}

}

}

}

catch(Exception e)

{

cntValid = 0;

// file read error

}

string[] keys = aafiles.keys; // workaround for bad codegen with files ~= aafiles.keys

files ~= keys;

sort(files); // for faster file access?

return cntValid > 0;

}

version(slow)

unittest

{

string line = r"std.file (c:[\\dmd\\phobos\\std\\file.d](file:///\\dmd\phobos\std\file.d)) : public : std.utf (c:[\\dmd\\phobos\\std\\utf.d](file:///\\dmd\phobos\std\utf.d))";

RegExp re = new RegExp(re\_match\_dep);

string[] match = re.exec(line);

assert(match.length == 3);

assert(match[0] == line);

assert(match[1] == r"c:[\\dmd\\phobos\\std\\file.d](file:///\\dmd\phobos\std\file.d)");

assert(match[2] == r"c:[\\dmd\\phobos\\std\\utf.d](file:///\\dmd\phobos\std\utf.d)");

line = r"std.file (c:[\\dmd\\phobos\\std\\file.d](file:///\\dmd\phobos\std\file.d)) : public : std.utf (c:[\\dmd\\phobos\\std\\utf.d):abc,def](file:///\\dmd\phobos\std\utf.d):abc,def)";

match = re.exec(line);

assert(match.length == 3);

assert(match[0] == line);

assert(match[1] == r"c:[\\dmd\\phobos\\std\\file.d](file:///\\dmd\phobos\std\file.d)");

assert(match[2] == r"c:[\\dmd\\phobos\\std\\utf.d](file:///\\dmd\phobos\std\utf.d)");

}

bool launchBuildPhobos(string workdir, string cmdfile, string cmdline, IVsOutputWindowPane pane)

{

/////////////

auto srpIVsLaunchPadFactory = queryService!(IVsLaunchPadFactory);

if (!srpIVsLaunchPadFactory)

return false;

scope(exit) release(srpIVsLaunchPadFactory);

ComPtr!(IVsLaunchPad) srpIVsLaunchPad;

HRESULT hr = srpIVsLaunchPadFactory.CreateLaunchPad(&srpIVsLaunchPad.ptr);

if(FAILED(hr) || !srpIVsLaunchPad.ptr)

return OutputErrorString(format("internal error: IVsLaunchPadFactory.CreateLaunchPad failed with rc=%x", hr));

try

{

std.file.write(cmdfile, cmdline);

}

catch(FileException e)

{

return OutputErrorString(format("internal error: cannot write file " ~ cmdfile ~ "\n"));

}

//                scope(exit) std.file.remove(cmdfile);

BSTR bstrOutput;

DWORD result;

hr = srpIVsLaunchPad.ExecCommand(/\* [in] LPCOLESTR pszApplicationName \*/ \_toUTF16z(getCmdPath()),

/\* [in] LPCOLESTR pszCommandLine \*/ \_toUTF16z("/Q /C " ~ quoteFilename(cmdfile)),

/\* [in] LPCOLESTR pszWorkingDir \*/ \_toUTF16z(workdir), // may be NULL, passed on to CreateProcess (wee Win32 API for details)

/\* [in] LAUNCHPAD\_FLAGS lpf \*/ LPF\_PipeStdoutToOutputWindow,

/\* [in] IVsOutputWindowPane \*pOutputWindowPane \*/ pane, // if LPF\_PipeStdoutToOutputWindow, which pane in the output window should the output be piped to

/\* [in] ULONG nTaskItemCategory \*/ 0, // if LPF\_PipeStdoutToTaskList is specified

/\* [in] ULONG nTaskItemBitmap \*/ 0, // if LPF\_PipeStdoutToTaskList is specified

/\* [in] LPCOLESTR pszTaskListSubcategory \*/ null, // "Build"w.ptr, // if LPF\_PipeStdoutToTaskList is specified

/\* [in] IVsLaunchPadEvents \*pVsLaunchPadEvents \*/ null, //pLaunchPadEvents,

/\* [out] DWORD \*pdwProcessExitCode \*/ &result,

/\* [out] BSTR \*pbstrOutput \*/ &bstrOutput); // all output generated (may be NULL)

return hr == S\_OK && result == 0;

}

// This file is part of Visual D

//

// Visual D integrates the D programming language into Visual Studio

// Copyright (c) 2010 by Rainer Schuetze, All Rights Reserved

//

// Distributed under the Boost Software License, Version 1.0.

// See accompanying file LICENSE\_1\_0.txt or copy at <http://www.boost.org/LICENSE_1_0.txt>

module visuald.chiernode;

import visuald.windows;

import std.string;

import std.path;

import std.utf;

import sdk.vsi.vsshell;

import sdk.vsi.vsshell80;

import sdk.vsi.vsshell110;

import sdk.vsi.ivssccmanager2;

import visuald.hierarchy;

import visuald.comutil;

import visuald.logutil;

import visuald.hierutil;

import visuald.stringutil;

import visuald.automation;

//import dproject;

enum ICON\_TYPE

{

ICON\_Open,

ICON\_Closed,

ICON\_StateImage

}

import visuald.chiercontainer;

const UINT IDMX\_NULLMENU = 0;

\_\_gshared CHierNode[VSITEMID] gVsItemMap;

\_\_gshared Object gVsItemMap\_sync;

\_\_gshared bool hierContainerIsSorted = true;

class CIVsTaskItemArray {}

class OpenDocumentList {}

//---------------------------------------------------------------------------

// Name: CHierNode

//

// Description:

// Base node class for every object in a hierarchy. Implements the idea of

// a node that has a parent, but no children.

//

// The project hierarchy does not need references to attached children and

// parents, as long as Dispose() does no harm to the integrity. The GC will

// take care of destruction.

//---------------------------------------------------------------------------

class CHierNode : DisposingDispatchObject

{

static void shared\_static\_this()

{

gVsItemMap\_sync = new Object;

}

this()

{

m\_grfStateFlags = ST\_DefaultFlags;

synchronized(gVsItemMap\_sync)

gVsItemMap[GetVsItemID()] = this;

logCall("added %x to gVsItemMap", GetVsItemID());

}

~this()

{

}

void removeFromItemMap(bool recurse)

{

synchronized(gVsItemMap\_sync)

gVsItemMap.remove(GetVsItemID());

logCall("removed %x from gVsItemMap", GetVsItemID());

}

override void Dispose()

{

//m\_extNode = release(m\_extNode);

}

static void setContainerIsSorted(bool sort)

{

synchronized(gVsItemMap\_sync)

{

hierContainerIsSorted = sort;

foreach(n; gVsItemMap)

{

if(auto c = cast(CHierContainer) n)

c.SetIsSortedList(sort);

}

}

}

public:

// IsKindOf checking

uint GetKindOf() { return 0; }

bool IsKindOf(uint hKind) { return (hKind == (hKind & GetKindOf())); }

// return a CHierNode typecasted to a VSITEMID or VISTEMID\_ROOT

// Override in each derived class that can also be a VSITEMID\_ROOT

VSITEMID GetVsItemID() { return cast(VSITEMID) cast(void\*) this; }

// return the itemid of the first child or VSITEMID\_NIL

VSITEMID GetFirstChildID(bool fDisplayOnly = true) { return VSITEMID\_NIL; }

// Used by the hierarchy in response to VSHPROPID\_FirstChild/VSHPROPID\_NextSibling. These

// properties are spec'd to only return member items (visible or not)

VSITEMID GetFirstMemberChildID() { return VSITEMID\_NIL; }

VSITEMID GetNextMemberSiblingID()

{

CHierNode pNode = m\_pNodeNext;

while(pNode && !pNode.IsMemberItem())

pNode = pNode.m\_pNodeNext;

return pNode ? pNode.GetVsItemID() : VSITEMID\_NIL;

}

// is this node expandable in the shell?

// should this node be auto expanded in the shell?

bool Expandable() { return false; }

bool ExpandByDefault() { return false; }

// is this node a container in the shell? it may be a container

// and still not be expandable if all child items are hidden.

bool IsContainer() { return false; }

// traverses to root node via parents

// the root node is expected to return the associated CVsHierarchy

CVsHierarchy GetCVsHierarchy()

{

if(!GetParent() || IsZombie())

return null;

CHierNode pNode = GetRootNode();

if(!pNode)

return null;

return pNode.GetCVsHierarchy();

}

//---------------------------------------------------------------------------

// Base-implementation of GetNestedHierarchy handles the failure case. Any

// node which will contain another hierarchy must over-ride this method.

//---------------------------------------------------------------------------

        int GetNestedHierarchy(in IID\* riid, void \*\*ppVsHierarchy, out VSITEMID pitemidNested)

{

return E\_FAIL;

}

uint GetContextMenu() { return IDMX\_NULLMENU; }

// all nodes which neeed to handle these functions should over-ride them

UINT GetIconIndex(ICON\_TYPE it)

{

assert(false, "You should be calling an over-ridden version of this...");

}

string GetCanonicalName()

{

string name = GetFullPath();

return toLower(name);

}

string GetFullPath()

{

return m\_strName;

}

int DoDefaultAction()

{

// each node which has a "default" action must over-ride

return S\_OK;

}

// Property functions

int GetProperty(VSHPROPID propid, out VARIANT pvar)

{

switch(propid)

{

case VSHPROPID\_Name:

case VSHPROPID\_Caption:

pvar.vt = VT\_BSTR;

// don't return a display caption longer than \_MAX\_PATH-1, since the tree control cannot

// handle it. instead, truncate the caption by ellipsing it (terminating it with "...").

pvar.bstrVal = allocBSTR(ellipseString(GetDisplayCaption(), \_MAX\_PATH));

return S\_OK;

case VSHPROPID\_IsNonMemberItem:

pvar.vt = VT\_BOOL;

pvar.boolVal = IsMemberItem();

return S\_OK;

case VSHPROPID\_Expanded:

pvar.vt = VT\_BOOL;

pvar.boolVal = IsExpanded();

return S\_OK;

case VSHPROPID\_AltHierarchy:

pvar.vt = VT\_UNKNOWN;

pvar.punkVal = addref(GetCVsHierarchy());

return S\_OK;

case VSHPROPID\_AltItemid:

pvar.vt = VT\_I4;

pvar.lVal = GetVsItemID();

return S\_OK;

case VSHPROPID\_IsNonSearchable:

pvar.vt = VT\_BOOL;

pvar.boolVal = false;

return S\_OK;

case VSHPROPID\_StateIconIndex:

pvar.vt = VT\_I4;

pvar.lVal = STATEICON\_NOSTATEICON;

if(IVsSccManager2 sccmgr = queryService!(SVsSccManager, IVsSccManager2)())

{

scope(exit) release(sccmgr);

auto path = \_toUTF16z(GetFullPath());

VsStateIcon icon;

DWORD sccStatus;

if(sccmgr.GetSccGlyph(1, &path, &icon, &sccStatus) == S\_OK)

pvar.lVal = icon;

}

return S\_OK;

case VSHPROPID\_OverlayIconIndex:

pvar.vt = VT\_I4;

pvar.lVal = OVERLAYICON\_NONE;

return S\_OK;

case VSHPROPID\_HasEnumerationSideEffects: // keep ankhsvn happy

case VSHPROPID\_ChildrenEnumerated:

pvar.vt = VT\_BOOL;

pvar.boolVal = false;

return S\_OK;

case VSHPROPID\_BrowseObject:

pvar.vt = VT\_UNKNOWN;

return QueryInterface(&IDispatch.iid, cast(void \*\*)&pvar.pdispVal);

case VSHPROPID\_ExtObject:

static if(!HideProjectItems)

{

pvar.vt = VT\_DISPATCH;

if(!m\_extNode)

m\_extNode = /\*addref\*/(newCom!ExtProjectItem(null, null, this));

pvar.pdispVal = addref(m\_extNode);

return S\_OK;

}

else

break;

case VSHPROPID\_TargetPlatformIdentifier:

pvar.vt = VT\_BSTR;

pvar.bstrVal = allocBSTR("Windows");

return S\_OK;

default:

break;

}

return DISP\_E\_MEMBERNOTFOUND;

}

int SetProperty(VSHPROPID propid, in VARIANT var)

{

switch(propid)

{

case VSHPROPID\_Expanded:

SetIsExpanded(var.boolVal != 0);

return S\_OK;

default:

break;

}

return DISP\_E\_MEMBERNOTFOUND;

}

// These three are to allow you to do something in response to the label

// edit. They just tell you the change in state, the changing of the name of your

// item will be handled through the SetProperty command. Note they track the

// following shell commands to our hierarchy:

// UIHWCMDID\_StartLabelEdit

// UIHWCMDID\_CommitLabelEdit

// UIHWCMDID\_CancelLabelEdit

// Defaults do nothing

int OnStartLabelEdit()

{

return E\_NOTIMPL;

}

int OnCommitLabelEdit()

{

return E\_NOTIMPL;

}

int OnCancelLabelEdit()

{

return E\_NOTIMPL;

}

// VSHPROPID.EditLabel

int GetEditLabel(BSTR \*ppEditLabel)

{

\*ppEditLabel = allocBSTR(GetName());

return S\_OK;

//return E\_NOTIMPL;

}

int SetEditLabel(in BSTR pEditLabel)

{

return E\_NOTIMPL;

}

// Task list support. Allows adding tasks to the passed in task array. Default does nothing

int GetTasks(CIVsTaskItemArray \*pTaskItemArray) { return S\_OK; }

// CHierNode Properties

string GetName()

{

return m\_strName;

}

void SetName(string newName, CVsHierarchy pCVsHierarchy = null)

{

m\_strName = newName;

if (pCVsHierarchy)

pCVsHierarchy.OnPropertyChanged(this, VSHPROPID\_Caption, 0);

}

// CHierNode Properties

// VSHPROPID\_Caption

string GetDisplayCaption()

{

return baseName(m\_strName);

}

// VSHPROPID\_Parent

CHierContainer GetParent() { return m\_pNodeParent; }

void SetParent(CHierContainer pNode) { m\_pNodeParent = pNode; }

// VSHPROPID\_NextSibling

void SetNext(CHierNode pHierNode) { m\_pNodeNext = pHierNode; }

CHierNode GetNext(bool fDisplayOnly = true)

{

CHierNode pNode = m\_pNodeNext;

if(fDisplayOnly)

while(pNode && !pNode.IsDisplayable())

pNode = pNode.m\_pNodeNext;

return pNode;

}

CHierNode GetHeadEx(bool fDisplayOnly = true) { return null; }

//---------------------------------------------------------------------------

// Gets the child of this object's parent that occurs directly before

// this node in the parent's list. This is an expensive operation, and is

// nowhere as straight-forward as getting the next child in this list.

//---------------------------------------------------------------------------

CHierNode GetPrev(bool fDisplayOnly = true)

{

if(GetParent())

return GetParent().GetPrevChildOf(this, fDisplayOnly);

return null;

}

// traverse the parent nodes and return node whose parent is NULL

CHierNode GetRootNode()

{

CHierNode pNode = this;

while(pNode.GetParent())

pNode = pNode.GetParent();

return pNode;

}

// is the given node an ancestor of this node

bool HasAncestor(CHierNode pNode)

{

CHierNode pAncestor = GetParent();

for(; pAncestor; pAncestor = pAncestor.GetParent())

if (pAncestor is pNode)

return true;

return false;

}

/+

int ExtExpand(EXPANDFLAGS expandflags, ref GUID rguidPersistenceSlot = GUID\_SolutionExplorer)

{

return E\_FAIL;

}

+/

//Is this node a zombie, with no Root node in it's hierarchy

bool IsZombie()

{

CHierNode pNode = GetRootNode();

if (!pNode) // || (VSITEMID\_ROOT != pNode.GetVsItemID()))

return true;

return false;

}

// Static helper which is used to detect stale itemid's for nodes which have

// gone away (different than the zombie case).

static bool IsValidCHierNode(VSITEMID itemid)

{

synchronized(gVsItemMap\_sync)

if(itemid in gVsItemMap)

return true;

return false;

}

// Updates UI

void ReDraw(bool bUpdateIcon = true, bool bUpdateStateIcon = true, bool bUpdateText = false)

{

// Root object or item must be in UI.

CHierContainer pParent = GetParent();

if(!pParent || pParent.HaveChildrenBeenEnumerated())

{

CVsHierarchy pHier = GetCVsHierarchy();

if(bUpdateIcon)

pHier.OnPropertyChanged(this, VSHPROPID\_IconIndex, 0);

if(bUpdateStateIcon)

pHier.OnPropertyChanged(this, VSHPROPID\_StateIconIndex, 0);

if(bUpdateText)

pHier.OnPropertyChanged(this, VSHPROPID\_Caption, 0);

}

}

int IsDocumentOpen(OpenDocumentList \*rgOpenDocuments = null) { return S\_FALSE; }

int CloseDocuments(bool bPromptToSave = false)

{

int hrRet = S\_OK;

assert(!IsZombie());

if(IsZombie())

return S\_OK;

/+

// We walk the RDT looking for all running documents attached to this hierarchy and itemid. There

// are cases where there may be two different editors (not views) open on the same document.

CComPtr<IEnumRunningDocuments> srpEnumRDT;

IVsRunningDocumentTable\* pRDT = \_VxModule.GetIVsRunningDocumentTable();

ASSERT(pRDT);

if(!pRDT)

return S\_OK;

HRESULT hr = pRDT.GetRunningDocumentsEnum(&srpEnumRDT);

ASSERT(SUCCEEDED(hr));

if(SUCCEEDED(hr))

{

VSCOOKIE dwDocCookie;

VSSLNCLOSEOPTIONS saveOptions = bPromptToSave? SLNSAVEOPT\_PromptSave : SLNSAVEOPT\_NoSave;

CComPtr<IVsHierarchy> srpOurHier = GetCVsHierarchy().GetIVsHierarchy();

srpEnumRDT.Reset();

while(srpEnumRDT.Next(1, &dwDocCookie, NULL) == S\_OK)

{

// Note we can pass NULL for all parameters we don't care about

CComPtr<IVsHierarchy> srpHier;

VSITEMID itemid = VSITEMID\_NIL;

pRDT.GetDocumentInfo(dwDocCookie, NULL/\*pgrfRDTFlags\*/, NULL/\*pdwReadLocks\*/, NULL/\*pdwEditLocks\*/,

NULL /\*bstrMkDocumentOld\*/, &srpHier, &itemid, NULL /\*ppunkDocData\*/);

// Is this one of our documents?

if(srpHier is srpOurHier && itemid == GetVsItemID())

{

// This is the only hr return code we care about

hrRet = \_VxModule.GetIVsSolution().CloseSolutionElement(saveOptions, srpOurHier, dwDocCookie);

if(FAILED(hrRet))

break;

}

}

}

+/

return hrRet;

}

int SaveDocument(bool bPromptToSave = false)

{

return E\_FAIL;

}

// VSHPROPID\_UserContext

int GetUserContext(IVsUserContext \*\*ppUserCtx)

{

return E\_NOTIMPL;

}

DWORD GetDisplayOrder() { return 0; }

HRESULT GetGuidProperty(VSHPROPID propid, out GUID pGuid)

{

return E\_NOTIMPL;

}

// IOleCommandTarget

public:

int QueryStatus(

/\* [unique][in] \*/ in GUID \*pguidCmdGroup,

/\* [in] \*/ ULONG cCmds,

/\* [out][in][size\_is] \*/ OLECMD\* prgCmds,

/\* [unique][out][in] \*/ OLECMDTEXT \*pCmdText)

{

//ATLTRACENOTIMPL(\_T("CHierNode::IOleCommandTarget::QueryStatus"));

return OLECMDERR\_E\_NOTSUPPORTED;

}

int Exec(

/\* [unique][in] \*/ in GUID \*pguidCmdGroup,

/\* [in] \*/ DWORD nCmdID,

/\* [in] \*/ DWORD nCmdexecopt,

/\* [unique][in] \*/ in VARIANT \*pvaIn,

/\* [unique][out][in] \*/ VARIANT \*pvaOut)

{

//ATLTRACENOTIMPL(\_T("CHierNode::IOleCommandTarget::Exec"));

return OLECMDERR\_E\_NOTSUPPORTED;

}

// Bit state functions

public:

void SetIsDisplayable(bool bValue) { SetBits(ST\_Displayable, bValue); }

bool IsDisplayable() { return IsSet(ST\_Displayable); }

void SetIsOpen(bool bValue) { SetBits(ST\_IsOpen, bValue); }

bool IsOpen() { return IsSet(ST\_IsOpen); }

void SetIsMemberItem(bool bValue) { SetBits(ST\_IsMemberItem, bValue); }

bool IsMemberItem() { return IsSet(ST\_IsMemberItem); }

void SetIsExpanded(bool bValue) { SetBits(ST\_Expanded, bValue); }

bool IsExpanded() { return IsSet(ST\_Expanded); }

//////////////////////////////////////////////////////////////

\_\_gshared ComTypeInfoHolder mTypeHolder;

static void shared\_static\_this\_typeHolder()

{

static class \_ComTypeInfoHolder : ComTypeInfoHolder

{

override int GetIDsOfNames(

/\* [size\_is][in] \*/ in LPOLESTR \*rgszNames,

/\* [in] \*/ in UINT cNames,

/\* [size\_is][out] \*/ MEMBERID \*pMemId)

{

//mixin(LogCallMix);

if (cNames == 1 && to\_string(\*rgszNames) == "\_\_id")

{

\*pMemId = 2;

return S\_OK;

}

return returnError(E\_NOTIMPL);

}

}

mTypeHolder = newCom!\_ComTypeInfoHolder;

addref(mTypeHolder);

}

static void shared\_static\_dtor\_typeHolder()

{

mTypeHolder = release(mTypeHolder);

}

override ComTypeInfoHolder getTypeHolder () { return mTypeHolder; }

//////////////////////////////////////////////////////////////

override int Invoke(

/\* [in] \*/ in DISPID dispIdMember,

/\* [in] \*/ in IID\* riid,

/\* [in] \*/ in LCID lcid,

/\* [in] \*/ in WORD wFlags,

/\* [out][in] \*/ DISPPARAMS \*pDispParams,

/\* [out] \*/ VARIANT \*pVarResult,

/\* [out] \*/ EXCEPINFO \*pExcepInfo,

/\* [out] \*/ UINT \*puArgErr)

{

mixin(LogCallMix);

if(dispIdMember == 1 || dispIdMember == 2)

{

if(pDispParams.cArgs == 0)

return GetProperty(VSHPROPID\_Name, \*pVarResult);

}

return returnError(E\_NOTIMPL);

}

//////////////////////////////////////////////////////////////

protected:

CHierContainer m\_pNodeParent;

CHierNode m\_pNodeNext; // to form a singly-linked list

string m\_strName; // this node's name

ExtProjectItem m\_extNode;

uint m\_grfStateFlags; // ChildrenEnumerated, etc

enum // m\_grfStateFlags

{

ST\_ChildrenEnumerated = (1<<0),

ST\_IsOpen = (1<<1), // File is open in an editor. Note user controlled. does not check

ST\_Displayable = (1<<2),

ST\_SortedList = (1<<3), // Containers only. True if a sorted by alpha list.

ST\_IsMemberItem = (1<<4), // true if this node is a member of the project

ST\_Expanded = (1<<5),

ST\_FirstUserFlag = (1<<8), // Derived classes are free to use these upper 24 bits

ST\_DefaultFlags = (ST\_Displayable | ST\_IsMemberItem)

}

// m\_grfStateFlags bit helpers

bool IsSet(int bits) { return (m\_grfStateFlags & bits) != 0; }

void SetBits(int bits, bool bValue)

{

if(bValue)

m\_grfStateFlags |= bits;

else

m\_grfStateFlags &= ~bits;

}

}

// This file is part of Visual D

//

// Visual D integrates the D programming language into Visual Studio

// Copyright (c) 2010 by Rainer Schuetze, All Rights Reserved

//

// Distributed under the Boost Software License, Version 1.0.

// See accompanying file LICENSE\_1\_0.txt or copy at <http://www.boost.org/LICENSE_1_0.txt>

module visuald.colorizer;

import visuald.windows;

import std.string;

import std.ascii;

import std.utf;

import std.conv;

import std.algorithm;

import std.datetime;

import visuald.comutil;

import visuald.logutil;

import visuald.hierutil;

import visuald.fileutil;

import visuald.stringutil;

import visuald.pkgutil;

import visuald.simpleparser;

import visuald.dpackage;

import visuald.dlangsvc;

import visuald.config;

import vdc.lexer;

import vdc.versions;

import stdext.string;

import sdk.port.vsi;

import sdk.vsi.textmgr;

import sdk.vsi.textmgr2;

import sdk.vsi.vsshell80;

// version = LOG;

enum TokenColor

{

// assumed to match lexer.TokenCat and colorableItems in dlangsvc.d

Text = cast(int) TokenCat.Text,

Keyword = TokenCat.Keyword,

Comment = TokenCat.Comment,

Identifier = TokenCat.Identifier,

String = TokenCat.String,

Literal = TokenCat.Literal,

Text2 = TokenCat.Text2,

Operator = TokenCat.Operator,

// colorizer specifics:

AsmRegister,

AsmMnemonic,

UserType,

Version,

DisabledKeyword,

DisabledComment,

DisabledIdentifier,

DisabledString,

DisabledLiteral,

DisabledText,

DisabledOperator,

DisabledAsmRegister,

DisabledAsmMnemonic,

DisabledUserType,

DisabledVersion,

StringKeyword,

StringComment,

StringIdentifier,

StringString,

StringLiteral,

StringText,

StringOperator,

StringAsmRegister,

StringAsmMnemonic,

StringUserType,

StringVersion,

CoverageKeyword,

NonCoverageKeyword,

}

int[wstring] parseUserTypes(string spec)

{

int color = TokenColor.UserType;

int[wstring] types;

types["\_\_ctfe"] = TokenColor.Keyword;

foreach(t; tokenizeArgs(spec))

{

switch(t)

{

case "[Keyword]":         color = TokenColor.Keyword;        break;

case "[Comment]":         color = TokenColor.Comment;        break;

case "[Identifier]": color = TokenColor.Identifier; break;

case "[String]":         color = TokenColor.String;                break;

case "[Number]":         color = TokenColor.Literal;        break;

case "[Text]":                 color = TokenColor.Text;                break;

case "[Operator]":         color = TokenColor.Operator;        break;

case "[Register]":         color = TokenColor.AsmRegister;break;

case "[Mnemonic]":         color = TokenColor.AsmMnemonic;break;

case "[Type]":                 color = TokenColor.UserType;        break;

case "[Version]":         color = TokenColor.Version;        break;

default: types[to!wstring(t)] = color; break;

}

}

return types;

}

///////////////////////////////////////////////////////////////////////////////

class ColorableItem : DComObject, IVsColorableItem, IVsHiColorItem

{

private string mDisplayName;

private COLORINDEX mBackground;

private COLORINDEX mForeground;

private COLORREF mRgbForeground;

private COLORREF mRgbBackground;

this(string displayName, COLORINDEX foreground, COLORINDEX background,

COLORREF rgbForeground = 0, COLORREF rgbBackground = 0)

{

mDisplayName = displayName;

mBackground = background;

mForeground = foreground;

mRgbForeground = rgbForeground;

mRgbBackground = rgbBackground;

}

override HRESULT QueryInterface(in IID\* riid, void\*\* pvObject)

{

if(queryInterface!(IVsColorableItem) (this, riid, pvObject))

return S\_OK;

if(queryInterface!(IVsHiColorItem) (this, riid, pvObject))

return S\_OK;

return super.QueryInterface(riid, pvObject);

}

// IVsColorableItem

HRESULT GetDefaultColors(/+[out]+/ COLORINDEX \*piForeground, /+[out]+/ COLORINDEX \*piBackground)

{

if(!piForeground || !piBackground)

return E\_INVALIDARG;

\*piForeground = mForeground;

\*piBackground = mBackground;

return S\_OK;

}

HRESULT GetDefaultFontFlags(/+[out]+/ DWORD \*pdwFontFlags) // see FONTFLAGS enum

{

if(!pdwFontFlags)

return E\_INVALIDARG;

\*pdwFontFlags = 0;

return S\_OK;

}

HRESULT GetDisplayName(/+[out]+/ BSTR \* pbstrName)

{

if(!pbstrName)

return E\_INVALIDARG;

\*pbstrName = allocBSTR(mDisplayName);

return S\_OK;

}

// IVsHiColorItem

HRESULT GetColorData(in VSCOLORDATA cdElement, /+[out]+/ COLORREF\* pcrColor)

{

if(cdElement == CD\_FOREGROUND && mForeground == -1)

{

\*pcrColor = mRgbForeground;

return S\_OK;

}

if(cdElement == CD\_BACKGROUND && mBackground == -1)

{

\*pcrColor = mRgbBackground;

return S\_OK;

}

return E\_NOTIMPL;

}

final HRESULT SetDefaultForegroundColor(COLORREF color)

{

mRgbForeground = color;

return S\_OK;

}

final string GetDisplayName()

{

return mDisplayName;

}

}

class Colorizer : DisposingComObject, IVsColorizer, ConfigModifiedListener

{

// mLineState keeps track of evaluated states, assuming the interesting lines have been processed

// after the last changes

// the lower 20 bits are used by the lexer, the upper 12 bits encode the version state

// TBBB\_BBBB\_PPPP

// PPPP - version parse state

// BBBB - brace count

// T - toggle bit to force change

int[] mLineState;

int mLastValidLine;

Source mSource;

ParserBase!wstring mParser;

Config mConfig;

bool mColorizeVersions;

bool mColorizeCoverage;

bool mParseSource;

enum int kIndexVersion = 0;

enum int kIndexDebug = 1;

// index 0 for version, index 1 for debug

int[wstring][2] mVersionIds; // positive: lineno defined

int[2] mVersionLevel = [ -1, -1 ];

int[2] mVersionLevelLine = [ -2, -2 ]; // -2 never defined, -1 if set on command line

int[wstring] mDebugIds; // positive: lineno defined

int mDebugLevel = -1;

int mDebugLevelLine = -2; // -2 never defined, -1 if set on command line

string[2] mConfigVersions;

ubyte mConfigRelease;

bool mConfigUnittest;

bool mConfigX64;

bool mConfigMSVCRT;

bool mConfigCoverage;

bool mConfigDoc;

bool mConfigNoBoundsCheck;

ubyte mConfigCompiler;

int[] mCoverage;

float mCoveragePercent;

string mLastCoverageFile;

SysTime mLastTestCoverageFile;

SysTime mLastModifiedCoverageFile;

enum VersionParseState

{

IdleEnabled,

IdleDisabled,

IdleEnabledVerify, // verify enable state on next token

IdleDisabledVerify,

VersionParsed, // version, expecting = or (

AssignParsed, // version=, expecting identifier or number

ParenLParsed, // version(, expecting identifier or number

IdentNumberParsedEnable, // version(identifier|number, expecting )

IdentNumberParsedDisable, // version(identifier|number, expecting )

ParenRParsedEnable, // version(identifier|number), check for '{'

ParenRParsedDisable, // version(identifier|number), check for '{'

AsmParsedEnabled, // enabled asm, expecting {

AsmParsedDisabled, // disabled asm, expecting {

InAsmBlockEnabled, // inside asm {}, expecting {

InAsmBlockDisabled, // inside disabled asm {}, expecting {

}

static assert(VersionParseState.max <= 15);

this(Source src)

{

mSource = src;

mParser = new ParserBase!wstring;

mColorizeVersions = Package.GetGlobalOptions().ColorizeVersions;

mColorizeCoverage = Package.GetGlobalOptions().ColorizeCoverage;

mParseSource = Package.GetGlobalOptions().parseSource;

UpdateConfig();

UpdateCoverage(true);

}

~this()

{

}

override void Dispose()

{

if(mConfig)

{

mConfig.RemoveModifiedListener(this);

mConfig = null;

}

}

override HRESULT QueryInterface(in IID\* riid, void\*\* pvObject)

{

if(queryInterface!(IVsColorizer) (this, riid, pvObject))

return S\_OK;

return super.QueryInterface(riid, pvObject);

}

// IVsColorizer //////////////////////////////////////

override int GetStateMaintenanceFlag(BOOL\* pfFlag)

{

// version(LOG) mixin(LogCallMix2);

\*pfFlag = false;

return S\_OK;

}

override int GetStartState(int\* piStartState)

{

version(LOG) mixin(LogCallMix2);

\*piStartState = 0;

return S\_OK;

}

override int ColorizeLine(in int iLine, in int iLength, in wchar\* pText, in int iState, uint\* pAttributes)

{

bool versionsChanged = false;

int state = GetLineState(iLine);

GetLineState(iLine + 1); // ensure the line has been parsed

wstring text = to\_cwstring(pText, iLength);

version(LOG) logCall("%s.ColorizeLine(%d,%x): %s", this, iLine, state, text);

version(LOG) mixin(\_LogIndentNoRet);

uint pos = 0;

bool inTokenString = (Lexer.tokenStringLevel(state) > 0);

int cov = -1;

int covtype = TokenColor.CoverageKeyword;

if(mColorizeCoverage && mCoverage.length)

{

int covLine = mSource.adjustLineNumberSinceLastBuildReverse(iLine, true);

cov = covLine >= mCoverage.length ? -1 : mCoverage[covLine];

covtype = cov == 0 ? TokenColor.NonCoverageKeyword : TokenColor.CoverageKeyword;

}

int back = 0; // COLOR\_MARKER\_MASK;

LanguageService langsvc = Package.GetLanguageService();

while(pos < iLength)

{

uint prevpos = pos;

int type = dLex.scan(state, text, pos);

bool nowInTokenString = (Lexer.tokenStringLevel(state) > 0);

wstring tok = text[prevpos..pos];

ParserSpan span;

if (pos >= text.length)

span = ParserSpan(prevpos, iLine, 0, iLine + 1);

else

span = ParserSpan(prevpos, iLine, pos, iLine);

if(tok[0] == 'i')

if(tok == "in" || tok == "is")

{

if(langsvc.isBinaryOperator(mSource, iLine + 1, prevpos, iLine + 1, pos))

type = TokenColor.Operator;

else

type = TokenColor.Keyword;

}

if(cov >= 0)

{

type = covtype;

}

else

{

if(mColorizeVersions)

{

if(Lexer.isCommentOrSpace(type, tok) || (inTokenString || nowInTokenString))

{

int parseState = getParseState(state);

if(type == TokenColor.Identifier || type == TokenColor.Keyword)

type = userColorType(tok, type);

if(parseState == VersionParseState.IdleDisabled || parseState == VersionParseState.IdleDisabledVerify)

type = disabledColorType(type);

}

else

{

type = parseVersions(span, type, tok, state, versionsChanged);

}

}

if(inTokenString || nowInTokenString)

type = stringColorType(type);

//else if(mParseSource)

//        type = parseErrors(span, type, tok);

}

inTokenString = nowInTokenString;

while(prevpos < pos)

pAttributes[prevpos++] = type | back;

}

pAttributes[iLength] = (cov >= 0 ? covtype : TokenColor.Text) | back;

return S\_OK;

}

override int GetStateAtEndOfLine(in int iLine, in int iLength, in wchar\* pText, in int iState)

{

version(LOG) mixin(LogCallMix2);

assert(\_false); // should not be called if GetStateMaintenanceFlag return false

bool versionsChanged;

wstring text = to\_cwstring(pText, iLength);

return GetStateAtEndOfLine(iLine, text, iState, versionsChanged);

}

int ScanAndParse(int iLine, wstring text, bool doShift, ref int state, ref uint pos, ref bool versionsChanged)

{

uint prevpos = pos;

int id;

int type = dLex.scan(state, text, pos, id);

if(mColorizeVersions)

{

wstring txt = text[prevpos..pos];

if(!dLex.isCommentOrSpace(type, txt))

{

ParserToken!wstring tok;

tok.type = type;

tok.text = txt;

tok.id = id;

if (pos >= text.length)

tok.span = ParserSpan(prevpos, iLine, 0, iLine + 1);

else

tok.span = ParserSpan(prevpos, iLine, pos, iLine);

bool inTokenString = (dLex.tokenStringLevel(state) > 0);

if(doShift)

mParser.shift(tok);

if (!inTokenString)

type = parseVersions(tok.span, type, txt, state, versionsChanged);

}

}

return type;

}

int GetStateAtEndOfLine(in int iLine, wstring text, in int iState, ref bool versionsChanged)

{

version(LOG) logCall("%s.GetStateAtEndOfLine(%d,%s,%x)", this, iLine, text, iState);

version(LOG) mixin(\_LogIndentNoRet);

// SaveLineState(iLine, iState);

if(mColorizeVersions)

{

versionsChanged = clearVersions(iLine);

syncParser(iLine);

}

int state = iState;

uint pos = 0;

while(pos < text.length)

ScanAndParse(iLine, text, true, state, pos, versionsChanged);

/\*

if(versionsChanged && iLine + 1 < mLineState.length)

{

int nextState = mLineState[iLine + 1];

if(nextState == state)

state ^= 1 << 31;

}

\*/

lastParserLine = iLine + 1;

lastParserIndex = 0;

version(LOG) logCall("%s.GetStateAtEndOfLine returns state %x", this, state);

return state;

}

override int CloseColorizer()

{

version(LOG) mixin(LogCallMix);

return S\_OK;

}

//////////////////////////////////////////////////////////////

void drawCoverageOverlay(HWND hwnd, WPARAM wParam, LPARAM lParam, IVsTextView view)

{

if(!mColorizeCoverage || !mCoverage.length)

return;

HDC hDC = GetDC(hwnd);

RECT r;

GetClientRect(hwnd, &r);

SelectObject(hDC, GetStockObject(BLACK\_PEN));

int h = 10;

view.GetLineHeight(&h);

int iMinUnit, iMaxUnit, iVisibleUnits, iFirstVisibleUnit;

view.GetScrollInfo (1, &iMinUnit, &iMaxUnit, &iVisibleUnits, &iFirstVisibleUnit);

LOGFONTW logfont;

FontInfo fontInfo;

ColorableItemInfo textColor, covColor, noncovColor, nocovColor;

IVsFontAndColorStorage pIVsFontAndColorStorage = queryService!(IVsFontAndColorStorage);

if(pIVsFontAndColorStorage)

{

scope(exit) release(pIVsFontAndColorStorage);

auto flags = FCSF\_READONLY|FCSF\_LOADDEFAULTS|FCSF\_NOAUTOCOLORS;

if(pIVsFontAndColorStorage.OpenCategory(&GUID\_TextEditorFC, flags) == S\_OK)

{

pIVsFontAndColorStorage.GetFont(&logfont, &fontInfo);

pIVsFontAndColorStorage.GetItem("Plain Text", &textColor);

pIVsFontAndColorStorage.GetItem("Indicator Margin", &covColor);

textColor.bBackgroundValid = covColor.bBackgroundValid;

textColor.crBackground = covColor.crBackground;

pIVsFontAndColorStorage.GetItem("Visual D Text Coverage", &covColor);

pIVsFontAndColorStorage.GetItem("Visual D Text Non-Coverage", &noncovColor);

pIVsFontAndColorStorage.GetItem("Visual D Margin No Coverage", &nocovColor);

pIVsFontAndColorStorage.CloseCategory();

}

}

HFONT fnt = CreateFontIndirect(&logfont);

if(fnt)

SelectObject(hDC, fnt);

SetTextAlign(hDC, TA\_RIGHT);

int x0 = r.right - 40;

for(int i = 0; i <= iVisibleUnits; i++)

{

RECT tr = { x0, h \* i, r.right, h \* i + h };

int iLine = iFirstVisibleUnit + i;

int covLine = mSource.adjustLineNumberSinceLastBuildReverse(iLine, true);

int cov = covLine >= mCoverage.length ? -1 : mCoverage[covLine];

string s;

ColorableItemInfo \*info = &covColor;

if(cov < 0)

{

if (iLine == 0 && mCoveragePercent >= 0)

s = text(mCoveragePercent) ~ "%";

info = &nocovColor;

}

else if(cov == 0)

{

s = "0";

info = &noncovColor;

}

else if(cov > 9999)

s = ">9999";

else

s = text(cov);

if(info.bForegroundValid)

SetTextColor(hDC, info.crForeground);

if(info.bBackgroundValid)

SetBkColor(hDC, info.crBackground);

ExtTextOutA(hDC, tr.right - 1, tr.top, ETO\_OPAQUE, &tr, s.ptr, s.length, null);

}

MoveToEx(hDC, x0, r.top, null);

LineTo(hDC, x0, r.bottom);

if(fnt)

DeleteObject(fnt);

ReleaseDC(hwnd, hDC);

}

//////////////////////////////////////////////////////////////

int lastParserLine;

int lastParserIndex;

void syncParser(int line)

{

if(line == lastParserLine && lastParserIndex == 0)

return;

lastParserLine = line;

lastParserIndex = 0;

mParser.prune(lastParserLine, lastParserIndex);

if(line == lastParserLine && lastParserIndex == 0)

return;

assert(lastParserLine >= 0 && lastParserLine < line);

assert(lastParserLine < mLineState.length);

version(LOG) logCall("%s.syncParser(%d) restarts at [%d,%d]", this, line, lastParserLine, lastParserIndex);

version(LOG) mixin(\_LogIndentNoRet);

int state = mLineState[lastParserLine];

assert(state != -1);

wstring text = mSource.GetText(lastParserLine, 0, lastParserLine, -1);

bool versionsChanged;

// scan until we find the position of the parser token

uint pos = 0;

while(pos < text.length && pos < lastParserIndex)

ScanAndParse(lastParserLine, text, false, state, pos, versionsChanged);

// parse the rest of the lines

for( ; ; )

{

while(pos < text.length)

ScanAndParse(lastParserLine, text, true, state, pos, versionsChanged);

lastParserLine++;

if(lastParserLine >= line)

break;

text = mSource.GetText(lastParserLine, 0, lastParserLine, -1);

pos = 0;

}

lastParserIndex = 0;

}

//////////////////////////////////////////////////////////////

bool \_clearVersions(int debugOrVersion, int iLine)

{

wstring[] toremove;

foreach(id, line; mVersionIds[debugOrVersion])

if(line == iLine)

toremove ~= id;

foreach(id; toremove)

mVersionIds[debugOrVersion].remove(id);

if(mVersionLevelLine[debugOrVersion] == iLine)

{

mVersionLevelLine[debugOrVersion] = -2;

mVersionLevel[debugOrVersion] = -1;

return true;

}

return toremove.length > 0;

}

bool clearVersions(int iLine)

{

return \_clearVersions(0, iLine)

| \_clearVersions(1, iLine);

}

void defineVersion(int line, int num, int debugOrVersion, ref bool versionsChanged)

{

if(mVersionLevel[debugOrVersion] < 0 || line < mVersionLevelLine[debugOrVersion])

{

mVersionLevelLine[debugOrVersion] = line;

mVersionLevel[debugOrVersion] = num;

versionsChanged = true;

}

}

bool isVersionEnabled(int line, int num, int debugOrVersion)

{

if(num == 0)

return true;

if(line >= mVersionLevelLine[debugOrVersion] && num <= mVersionLevel[debugOrVersion])

return true;

string versionids = mConfigVersions[debugOrVersion];

string[] versions = tokenizeArgs(versionids);

foreach(ver; versions)

if(dLex.isInteger(ver) && to!int(ver) >= num)

return true;

return false;

}

bool defineVersion(int line, wstring ident, int debugOrVersion, ref bool versionsChanged)

{

if (debugOrVersion == 0)

{

int res = versionPredefined(ident);

if(res != 0)

return false;

}

int \*pline = ident in mVersionIds[debugOrVersion];

if(!pline)

mVersionIds[debugOrVersion][ident] = line;

else if(\*pline < 0 && -\*pline > line)

\*pline = line;

else if(\*pline >= 0 && \*pline > line)

\*pline = line;

else if(\*pline >= 0 && \*pline == line)

return true;

else

return false;

versionsChanged = true;

return true;

}

\_\_gshared int[wstring] predefinedVersions;

shared static this()

{

foreach(v, p; vdc.versions.sPredefinedVersions)

predefinedVersions[to!wstring(v)] = p;

}

int versionPredefined(wstring ident)

{

int\* p = ident in predefinedVersions;

if(!p)

return 0;

if(\*p != 0)

return \*p;

switch(ident)

{

case "unittest":

return mConfigUnittest ? 1 : -1;

case "assert":

return mConfigUnittest || mConfigRelease != 1 ? 1 : -1;

case "D\_Coverage":

return mConfigCoverage ? 1 : -1;

case "D\_Ddoc":

return mConfigDoc ? 1 : -1;

case "D\_NoBoundsChecks":

return mConfigNoBoundsCheck ? 1 : -1;

case "Win32":

case "X86":

case "D\_InlineAsm\_X86":

return mConfigX64 ? -1 : 1;

case "Win64":

case "X86\_64":

case "D\_InlineAsm\_X86\_64":

case "D\_LP64":

return mConfigX64 ? 1 : -1;

case "GNU":

return mConfigCompiler == Compiler.GDC ? 1 : -1;

case "LDC":

return mConfigCompiler == Compiler.LDC ? 1 : -1;

case "DigitalMars":

return mConfigCompiler == Compiler.DMD ? 1 : -1;

case "CRuntime\_DigitalMars":

return mConfigCompiler == Compiler.DMD && !mConfigMSVCRT ? 1 : -1;

case "CRuntime\_Microsoft":

return (mConfigCompiler == Compiler.DMD || mConfigCompiler == Compiler.LDC) && mConfigMSVCRT ? 1 : -1;

case "MinGW":

return mConfigCompiler == Compiler.GDC || (mConfigCompiler == Compiler.LDC && !mConfigMSVCRT) ? 1 : -1;

default:

assert(false, "inconsistent predefined versions");

}

}

bool isVersionEnabled(int line, wstring ident, int debugOrVersion)

{

if(dLex.isInteger(ident))

return isVersionEnabled(line, to!int(ident), debugOrVersion);

if (debugOrVersion)

{

if(ident.length == 0 && mConfigRelease != 0)

return true;

}

else

{

int res = versionPredefined(ident);

if(res < 0)

return false;

if(res > 0)

return true;

}

string versionids = mConfigVersions[debugOrVersion];

string[] versions = tokenizeArgs(versionids);

foreach(ver; versions)

if(cmp(ver, ident) == 0)

return true;

int \*pline = ident in mVersionIds[debugOrVersion];

if(!pline || \*pline < 0 || \*pline > line)

return false;

return true;

}

int disabledColorType(int type)

{

switch(type)

{

case TokenColor.Text2:

case TokenColor.Text: return TokenColor.DisabledText;

case TokenColor.Keyword: return TokenColor.DisabledKeyword;

case TokenColor.Comment: return TokenColor.DisabledComment;

case TokenColor.Identifier: return TokenColor.DisabledIdentifier;

case TokenColor.String: return TokenColor.DisabledString;

case TokenColor.Literal: return TokenColor.DisabledLiteral;

case TokenColor.Operator: return TokenColor.DisabledOperator;

case TokenColor.AsmRegister: return TokenColor.DisabledAsmRegister;

case TokenColor.AsmMnemonic: return TokenColor.DisabledAsmMnemonic;

case TokenColor.UserType: return TokenColor.DisabledUserType;

case TokenColor.Version: return TokenColor.DisabledVersion;

default: break;

}

return type;

}

int stringColorType(int type)

{

switch(type)

{

case TokenColor.Text2:

case TokenColor.Text: return TokenColor.StringText;

case TokenColor.Keyword: return TokenColor.StringKeyword;

case TokenColor.Comment: return TokenColor.StringComment;

case TokenColor.Identifier: return TokenColor.StringIdentifier;

case TokenColor.String: return TokenColor.StringString;

case TokenColor.Literal: return TokenColor.StringLiteral;

case TokenColor.AsmRegister: return TokenColor.StringAsmRegister;

case TokenColor.AsmMnemonic: return TokenColor.StringAsmMnemonic;

case TokenColor.UserType: return TokenColor.StringUserType;

case TokenColor.Version: return TokenColor.StringVersion;

default: break;

}

return type;

}

\_\_gshared int[wstring] asmIdentifiers;

static const wstring[] asmKeywords = [ "\_\_LOCAL\_SIZE", "dword", "even", "far", "naked", "near", "ptr", "qword", "seg", "word", ];

static const wstring[] asmRegisters = [

"AL", "AH", "AX", "EAX",

"BL", "BH", "BX", "EBX",

"CL", "CH", "CX", "ECX",

"DL", "DH", "DX", "EDX",

"BP", "EBP", "SP", "ESP",

"DI", "EDI", "SI", "ESI",

"ES", "CS", "SS", "DS", "GS", "FS",

"CR0", "CR2", "CR3", "CR4",

"DR0", "DR1", "DR2", "DR3", "DR4", "DR5", "DR6", "DR7",

"TR3", "TR4", "TR5", "TR6", "TR7",

"MM0", "MM1", "MM2", "MM3", "MM4", "MM5", "MM6", "MM7",

"XMM0", "XMM1", "XMM2", "XMM3", "XMM4", "XMM5", "XMM6", "XMM7",

];

static const wstring[] asmMnemonics = [

"\_\_emit", "\_emit", "aaa", "aad", "aam", "aas",

"adc", "add", "addpd", "addps", "addsd", "addss",

"addsubpd", "addsubps", "and", "andnpd", "andnps", "andpd",

"andps", "arpl", "blendpd", "blendps", "blendvpd", "blendvps",

"bound", "bsf", "bsr", "bswap", "bt", "btc",

"btr", "bts", "call", "cbw", "cdq", "cdqe",

"clc", "cld", "clflush", "cli", "clts", "cmc",

"cmova", "cmovae", "cmovb", "cmovbe", "cmovc", "cmove",

"cmovg", "cmovge", "cmovl", "cmovle", "cmovna", "cmovnae",

"cmovnb", "cmovnbe", "cmovnc", "cmovne", "cmovng", "cmovnge",

"cmovnl", "cmovnle", "cmovno", "cmovnp", "cmovns", "cmovnz",

"cmovo", "cmovp", "cmovpe", "cmovpo", "cmovs", "cmovz",

"cmp", "cmppd", "cmpps", "cmps", "cmpsb", "cmpsd",

"cmpsq", "cmpss", "cmpsw", "cmpxchg", "cmpxchg16b", "cmpxchg8b",

"comisd", "comiss", "cpuid", "cqo", "crc32", "cvtdq2pd",

"cvtdq2ps", "cvtpd2dq", "cvtpd2pi", "cvtpd2ps", "cvtpi2pd", "cvtpi2ps",

"cvtps2dq", "cvtps2pd", "cvtps2pi", "cvtsd2si", "cvtsd2ss", "cvtsi2sd",

"cvtsi2ss", "cvtss2sd", "cvtss2si", "cvttpd2dq", "cvttpd2pi", "cvttps2dq",

"cvttps2pi", "cvttsd2si", "cvttss2si", "cwd", "cwde", "da",

"daa", "das", "db", "dd", "de", "dec",

"df", "di", "div", "divpd", "divps", "divsd",

"divss", "dl", "dppd", "dpps", "dq", "ds",

"dt", "dw", "emms", "enter", "extractps", "f2xm1",

"fabs", "fadd", "faddp", "fbld", "fbstp", "fchs",

"fclex", "fcmovb", "fcmovbe", "fcmove", "fcmovnb", "fcmovnbe",

"fcmovne", "fcmovnu", "fcmovu", "fcom", "fcomi", "fcomip",

"fcomp", "fcompp", "fcos", "fdecstp", "fdisi", "fdiv",

"fdivp", "fdivr", "fdivrp", "feni", "ffree", "fiadd",

"ficom", "ficomp", "fidiv", "fidivr", "fild", "fimul",

"fincstp", "finit", "fist", "fistp", "fisttp", "fisub",

"fisubr", "fld", "fld1", "fldcw", "fldenv", "fldl2e",

"fldl2t", "fldlg2", "fldln2", "fldpi", "fldz", "fmul",

"fmulp", "fnclex", "fndisi", "fneni", "fninit", "fnop",

"fnsave", "fnstcw", "fnstenv", "fnstsw", "fpatan", "fprem",

"fprem1", "fptan", "frndint", "frstor", "fsave", "fscale",

"fsetpm", "fsin", "fsincos", "fsqrt", "fst", "fstcw",

"fstenv", "fstp", "fstsw", "fsub", "fsubp", "fsubr",

"fsubrp", "ftst", "fucom", "fucomi", "fucomip", "fucomp",

"fucompp", "fwait", "fxam", "fxch", "fxrstor", "fxsave",

"fxtract", "fyl2x", "fyl2xp1", "haddpd", "haddps", "hlt",

"hsubpd", "hsubps", "idiv", "imul", "in", "inc",

"ins", "insb", "insd", "insertps", "insw", "int",

"into", "invd", "invlpg", "iret", "iretd", "ja",

"jae", "jb", "jbe", "jc", "jcxz", "je",

"jecxz", "jg", "jge", "jl", "jle", "jmp",

"jna", "jnae", "jnb", "jnbe", "jnc", "jne",

"jng", "jnge", "jnl", "jnle", "jno", "jnp",

"jns", "jnz", "jo", "jp", "jpe", "jpo",

"js", "jz", "lahf", "lar", "lddqu", "ldmxcsr",

"lds", "lea", "leave", "les", "lfence", "lfs",

"lgdt", "lgs", "lidt", "lldt", "lmsw", "lock",

"lods", "lodsb", "lodsd", "lodsq", "lodsw", "loop",

"loope", "loopne", "loopnz", "loopz", "lsl", "lss",

"ltr", "maskmovdqu", "maskmovq", "maxpd", "maxps", "maxsd",

"maxss", "mfence", "minpd", "minps", "minsd", "minss",

"monitor", "mov", "movapd", "movaps", "movd", "movddup",

"movdq2q", "movdqa", "movdqu", "movhlps", "movhpd", "movhps",

"movlhps", "movlpd", "movlps", "movmskpd", "movmskps", "movntdq",

"movntdqa", "movnti", "movntpd", "movntps", "movntq", "movq",

"movq2dq", "movs", "movsb", "movsd", "movshdup", "movsldup",

"movsq", "movss", "movsw", "movsx", "movupd", "movups",

"movzx", "mpsadbw", "mul", "mulpd", "mulps", "mulsd",

"mulss", "mwait", "neg", "nop", "not", "or",

"orpd", "orps", "out", "outs", "outsb", "outsd",

"outsw", "pabsb", "pabsd", "pabsw", "packssdw", "packsswb",

"packusdw", "packuswb", "paddb", "paddd", "paddq", "paddsb",

"paddsw", "paddusb", "paddusw", "paddw", "palignr", "pand",

"pandn", /\*"pause",\*/ "pavgb", "pavgusb", "pavgw", "pblendvb",

"pblendw", "pcmpeqb", "pcmpeqd", "pcmpeqq", "pcmpeqw", "pcmpestri",

"pcmpestrm", "pcmpgtb", "pcmpgtd", "pcmpgtq", "pcmpgtw", "pcmpistri",

"pcmpistrm", "pextrb", "pextrd", "pextrq", "pextrw", "pf2id",

"pfacc", "pfadd", "pfcmpeq", "pfcmpge", "pfcmpgt", "pfmax",

"pfmin", "pfmul", "pfnacc", "pfpnacc", "pfrcp", "pfrcpit1",

"pfrcpit2", "pfrsqit1", "pfrsqrt", "pfsub", "pfsubr", "phaddd",

"phaddsw", "phaddw", "phminposuw", "phsubd", "phsubsw", "phsubw",

"pi2fd", "pinsrb", "pinsrd", "pinsrq", "pinsrw", "pmaddubsw",

"pmaddwd", "pmaxsb", "pmaxsd", "pmaxsw", "pmaxub", "pmaxud",

"pmaxuw", "pminsb", "pminsd", "pminsw", "pminub", "pminud",

"pminuw", "pmovmskb", "pmovsxbd", "pmovsxbq", "pmovsxbw", "pmovsxdq",

"pmovsxwd", "pmovsxwq", "pmovzxbd", "pmovzxbq", "pmovzxbw", "pmovzxdq",

"pmovzxwd", "pmovzxwq", "pmuldq", "pmulhrsw", "pmulhrw", "pmulhuw",

"pmulhw", "pmulld", "pmullw", "pmuludq", "pop", "popa",

"popad", "popcnt", "popf", "popfd", "popfq", "por",

"prefetchnta","prefetcht0", "prefetcht1", "prefetcht2", "psadbw", "pshufb",

"pshufd", "pshufhw", "pshuflw", "pshufw", "psignb", "psignd",

"psignw", "pslld", "pslldq", "psllq", "psllw", "psrad",

"psraw", "psrld", "psrldq", "psrlq", "psrlw", "psubb",

"psubd", "psubq", "psubsb", "psubsw", "psubusb", "psubusw",

"psubw", "pswapd", "ptest", "punpckhbw", "punpckhdq", "punpckhqdq",

"punpckhwd", "punpcklbw", "punpckldq", "punpcklqdq", "punpcklwd", "push",

"pusha", "pushad", "pushf", "pushfd", "pushfq", "pxor",

"rcl", "rcpps", "rcpss", "rcr", "rdmsr", "rdpmc",

"rdtsc", "rep", "repe", "repne", "repnz", "repz",

"ret", "retf", "rol", "ror", "roundpd", "roundps",

"roundsd", "roundss", "rsm", "rsqrtps", "rsqrtss", "sahf",

"sal", "sar", "sbb", "scas", "scasb", "scasd",

"scasq", "scasw", "seta", "setae", "setb", "setbe",

"setc", "sete", "setg", "setge", "setl", "setle",

"setna", "setnae", "setnb", "setnbe", "setnc", "setne",

"setng", "setnge", "setnl", "setnle", "setno", "setnp",

"setns", "setnz", "seto", "setp", "setpe", "setpo",

"sets", "setz", "sfence", "sgdt", "shl", "shld",

"shr", "shrd", "shufpd", "shufps", "sidt", "sldt",

"smsw", "sqrtpd", "sqrtps", "sqrtsd", "sqrtss", "stc",

"std", "sti", "stmxcsr", "stos", "stosb", "stosd",

"stosq", "stosw", "str", "sub", "subpd", "subps",

"subsd", "subss", "syscall", "sysenter", "sysexit", "sysret",

"test", "ucomisd", "ucomiss", "ud2", "unpckhpd", "unpckhps",

"unpcklpd", "unpcklps", "verr", "verw", "wait", "wbinvd",

"wrmsr", "xadd", "xchg", "xlat", "xlatb", "xor",

"xorpd", "xorps",

];

shared static this()

{

foreach(id; asmKeywords)

asmIdentifiers[id] = TokenColor.Keyword;

foreach(id; asmRegisters)

asmIdentifiers[id] = TokenColor.AsmRegister;

foreach(id; asmMnemonics)

asmIdentifiers[id] = TokenColor.AsmMnemonic;

}

private int asmColorType(wstring text)

{

if(auto p = text in asmIdentifiers)

return \*p;

return TokenColor.Identifier;

}

private int userColorType(wstring text, int type)

{

if(auto p = text in Package.GetGlobalOptions().UserTypes)

return \*p;

return type;

}

private static int getParseState(int iState)

{

return (iState >> 20) & 0x0f;

}

private static int getDebugOrVersion(int iState)

{

return (iState >> 24) & 0x1;

}

int parseVersions(ref ParserSpan span, int type, wstring text, ref int iState, ref bool versionsChanged)

{

int iLine = span.iStartLine;

version(none)

{

// COLORIZER\_ATTRIBUTE flags

// 0x00100: gray background

// 0x00200: black on dark blue

// 0x40000: underlined

// COLOR\_MARKER\_MASK = 0x00003f00: select color encoding, 0 standard, other from color list

// LINE\_MARKER\_MASK = 0x000fc000: underline style: 0-none, 4~blue, 5~red, 6~magenta, 7-gray, 11~green,

// 16-black, 23=magenta, 24=red, 35-maroon, 56-yellow, 58-ltgray

// PRIVATE\_CLIENT\_MASK1 = 0x00100000:

// PRIVATE\_CLIENT\_MASK2 = 0x00600000: ident marker style: 0-none, 1-blue start mark, 2,3-red end mark

// PRIVATE\_CLIENT\_MASK3 = 0x00800000: disable text coloring

// PRIVATE\_EDITOR\_MASK = 0xfc000000:

// SEPARATOR\_AFTER\_ATTR = 0x02000000: if on char after line, draws line between text rows

int lineMarker = 0; // iLine & 0x3f;

int privClient = (iLine >> 0) & 0xf;

int privEditor = (iLine >> 4) & 0x3f;

int attr = (lineMarker << 14) | (privClient << 20) | (privEditor << 26);

type |= attr;

}

version(all)

{

//if(dLex.isCommentOrSpace(type, text))

//        return type;

int parseState = getParseState(iState);

int debugOrVersion = getDebugOrVersion(iState);

int ntype = type;

if(ntype == TokenColor.Identifier || ntype == TokenColor.Keyword)

ntype = userColorType(text, ntype);

final switch(cast(VersionParseState) parseState)

{

case VersionParseState.IdleDisabledVerify:

case VersionParseState.IdleEnabledVerify:

if(isAddressEnabled(span.iStartLine, span.iStartIndex))

{

parseState = VersionParseState.IdleEnabled;

goto case VersionParseState.IdleEnabled;

}

parseState = VersionParseState.IdleDisabled;

goto case VersionParseState.IdleDisabled;

case VersionParseState.IdleDisabled:

ntype = disabledColorType(ntype);

if(text == "asm")

parseState = VersionParseState.AsmParsedDisabled;

else if(versionPredefined(text) && isVersionCondition(span))

ntype = TokenColor.DisabledVersion;

break;

case VersionParseState.IdleEnabled:

if(text == "version")

{

parseState = VersionParseState.VersionParsed;

debugOrVersion = 0;

}

else if(text == "debug")

{

parseState = VersionParseState.VersionParsed;

debugOrVersion = 1;

}

else if(text == "asm")

parseState = VersionParseState.AsmParsedEnabled;

break;

case VersionParseState.VersionParsed:

if(text == "=")

parseState = VersionParseState.AssignParsed;

else if(text == "(")

parseState = VersionParseState.ParenLParsed;

else if(debugOrVersion)

{

if(isVersionEnabled(iLine, "", debugOrVersion))

{

parseState = VersionParseState.IdleEnabled;

goto case VersionParseState.IdleEnabled;

}

else

{

parseState = VersionParseState.IdleDisabled;

goto case VersionParseState.IdleDisabled;

}

}

else

parseState = VersionParseState.IdleEnabled;

break;

case VersionParseState.AssignParsed:

if(dLex.isIdentifier(text))

{

if(debugOrVersion == 0 && versionPredefined(text))

ntype = TokenColor.Version;

if(!defineVersion(iLine, text, debugOrVersion, versionsChanged))

ntype |= 5 << 14; // red ~~~~ on VS2008

}

else if(dLex.isInteger(text))

defineVersion(iLine, to!int(text), debugOrVersion, versionsChanged);

parseState = VersionParseState.IdleEnabled;

break;

case VersionParseState.ParenLParsed:

if(dLex.isIdentifier(text) || dLex.isInteger(text))

{

if(debugOrVersion == 0 && versionPredefined(text))

ntype = TokenColor.Version;

if(isVersionEnabled(iLine, text, debugOrVersion))

parseState = VersionParseState.IdentNumberParsedEnable;

else

parseState = VersionParseState.IdentNumberParsedDisable;

}

else

parseState = VersionParseState.IdleEnabled;

break;

case VersionParseState.IdentNumberParsedDisable:

if(text == ")")

parseState = VersionParseState.ParenRParsedDisable;

else

parseState = VersionParseState.IdleEnabled;

break;

case VersionParseState.IdentNumberParsedEnable:

if(text == ")")

parseState = VersionParseState.ParenRParsedEnable;

else

parseState = VersionParseState.IdleEnabled;

break;

case VersionParseState.ParenRParsedEnable:

parseState = VersionParseState.IdleEnabled;

goto case VersionParseState.IdleEnabled;

case VersionParseState.ParenRParsedDisable:

parseState = VersionParseState.IdleDisabled;

goto case VersionParseState.IdleDisabled;

// asm block

case VersionParseState.AsmParsedEnabled:

if(text == "{")

parseState = VersionParseState.InAsmBlockEnabled;

else

parseState = VersionParseState.IdleEnabled;

break;

case VersionParseState.AsmParsedDisabled:

if(text == "{")

parseState = VersionParseState.InAsmBlockDisabled;

else

parseState = VersionParseState.IdleDisabled;

goto case VersionParseState.IdleDisabled;

case VersionParseState.InAsmBlockEnabled:

if(text == "}")

parseState = VersionParseState.IdleEnabled;

else if(ntype == TokenColor.Identifier)

ntype = asmColorType(text);

break;

case VersionParseState.InAsmBlockDisabled:

if(text == "}")

parseState = VersionParseState.IdleDisabled;

else if(ntype == TokenColor.Identifier)

ntype = asmColorType(text);

goto case VersionParseState.IdleDisabled;

}

if(text == ";" || text == "}")

{

if(parseState == VersionParseState.IdleDisabled)

parseState = text == ";" ? VersionParseState.IdleDisabledVerify : VersionParseState.IdleEnabledVerify;

}

else if(text == "else")

{

if(isAddressEnabled(span.iEndLine, span.iEndIndex))

{

parseState = VersionParseState.IdleEnabled;

ntype = type; // restore enabled type

}

else

{

parseState = VersionParseState.IdleDisabled;

ntype = disabledColorType(ntype);

}

}

iState = (iState & 0x800fffff) | (parseState << 20) | (debugOrVersion << 24);

}

return ntype;

}

int parseErrors(ref ParserSpan span, int type, wstring tok)

{

if(!dLex.isCommentOrSpace(type, tok))

if(mSource.hasParseError(span))

type |= 5 << 14; // red ~

return type;

}

wstring getVersionToken(LocationBase!wstring verloc)

{

if(verloc.children.length == 0)

return "";

ParserSpan span = verloc.children[0].span;

wstring text = mSource.GetText(span.iStartLine, span.iStartIndex, span.iEndLine, span.iEndIndex);

text = strip(text);

if(text.length == 0 || text[0] != '(' || text[$-1] != ')')

return ""; // parsing unfinished or debug statement without argument

text = strip(text[1..$-1]);

return text;

}

bool isAddressEnabled(int iLine, int iIndex)

{

mParser.fixExtend();

LocationBase!wstring loc = mParser.findLocation(iLine, iIndex, true);

LocationBase!wstring child = null;

while(loc)

{

if(VersionStatement!wstring verloc = cast(VersionStatement!wstring) loc)

{

wstring ver = getVersionToken(verloc);

if(isVersionEnabled(verloc.span.iStartLine, ver, 0))

{

if(verloc.children.length > 2 && child == verloc.children[2]) // spanContains(verloc.children[2].span, iLine, iIndex))

return false; // else statement

}

else

{

if(verloc.children.length > 1 && child == verloc.children[1]) // spanContains(verloc.children[1].span, iLine, iIndex))

return false; // then statement

}

}

else if(DebugStatement!wstring dbgloc = cast(DebugStatement!wstring) loc)

{

wstring ver = getVersionToken(dbgloc);

if(isVersionEnabled(dbgloc.span.iStartLine, ver, 1))

{

if(dbgloc.children.length > 2 && child == dbgloc.children[2]) // spanContains(dbgloc.children[2].span, iLine, iIndex))

return false; // else statement

}

else

{

if(dbgloc.children.length > 1 && child == dbgloc.children[1]) // spanContains(dbgloc.children[1].span, iLine, iIndex))

return false; // then statement

}

}

child = loc;

loc = loc.parent;

}

return true;

}

bool isVersionCondition(ref ParserSpan vspan)

{

mParser.fixExtend();

LocationBase!wstring loc = mParser.findLocation(vspan.iStartLine, vspan.iStartIndex, true);

LocationBase!wstring child = null;

while(loc)

{

if(VersionStatement!wstring verloc = cast(VersionStatement!wstring) loc)

{

if(verloc.children.length > 0)

{

if(spanContains(verloc.children[0].span, vspan.iStartLine, vspan.iStartIndex))

return true;

}

}

child = loc;

loc = loc.parent;

}

return false;

}

bool isInUnittest(int iLine, int iIndex)

{

mParser.fixExtend();

LocationBase!wstring loc = mParser.findLocation(iLine, iIndex, true);

LocationBase!wstring child = null;

while(loc)

{

if(auto utloc = cast(UnittestStatement!wstring) loc)

return true;

child = loc;

loc = loc.parent;

}

return false;

}

//////////////////////////////////////////////////////////////

void SaveLineState(int iLine, int state)

{

if(iLine >= mLineState.length)

{

int i = mLineState.length;

mLineState.length = iLine + 100;

for( ; i < mLineState.length; i++)

mLineState[i] = -1;

}

mLineState[iLine] = state;

}

void UpdateLineState(int line)

{

UpdateLineStates(line, line);

}

void UpdateLineStates(int line, int endline)

{

version(LOG) mixin(LogCallMix2);

int ln = line;

if(ln >= mLineState.length)

ln = max(mLineState.length, 1) - 1;

while(ln > 0 && mLineState[ln] == -1)

ln--;

if(ln == 0)

SaveLineState(0, 0);

int state = mLineState[ln];

bool stateChanged = false;

bool versionsChanged = false;

while(ln <= endline)

{

SaveLineState(ln, state);

wstring txt = mSource.GetText(ln, 0, ln, -1);

state = GetStateAtEndOfLine(ln, txt, state, versionsChanged);

ln++;

}

int prevState = ln < mLineState.length ? mLineState[ln] : -1;

SaveLineState(ln, state);

if(versionsChanged || mColorizeVersions || state != prevState)

{

ln++;

while(ln < mLineState.length)

{

if(mLineState[ln] == -1)

break;

mLineState[ln++] = -1;

}

mSource.ReColorizeLines(line, -1);

}

}

int GetLineState(int iLine)

{

int state = -1;

if(iLine >= 0 && iLine < mLineState.length)

state = mLineState[iLine];

if(state == -1)

{

UpdateLineState(iLine);

state = mLineState[iLine];

}

assert(state != -1);

return state;

}

//////////////////////////////////////////////////////////////

int OnLinesChanged(int iStartLine, int iOldEndLine, int iNewEndLine, bool fLast)

{

version(LOG) mixin(LogCallMix);

int p;

int diffLines = iNewEndLine - iOldEndLine;

int lines = mSource.GetLineCount(); // new line count

SaveLineState(lines, -1); // ensure mLineState[] is large enough

if(diffLines > 0)

{

for(p = lines; p > iNewEndLine; p--)

mLineState[p] = mLineState[p - diffLines];

for(; p > iStartLine; p--)

mLineState[p] = -1;

}

else if(diffLines < 0)

{

for(p = iStartLine + 1; p < iNewEndLine; p++)

mLineState[p] = -1;

for(; p - diffLines < lines; p++)

mLineState[p] = mLineState[p - diffLines];

for(; p < lines; p++)

mLineState[p] = -1;

}

if(iStartLine < mLineState.length && mLineState[iStartLine] != -1)

UpdateLineStates(iStartLine, iNewEndLine);

return S\_OK;

}

//////////////////////////////////////////////////////////////

int modifyValue(V)(V val, ref V var)

{

if(var == val)

return 0;

var = val;

return 1;

}

bool UpdateConfig()

{

int changes = 0;

string file = mSource.GetFileName ();

Config cfg = getProjectConfig(file);

release(cfg); // we don't need a reference

if(cfg != mConfig)

{

if(mConfig)

mConfig.RemoveModifiedListener(this);

mConfig = cfg;

if(mConfig)

mConfig.AddModifiedListener(this);

changes++;

}

if(mConfig)

{

ProjectOptions opts = mConfig.GetProjectOptions();

changes += modifyValue(opts.versionids, mConfigVersions[kIndexVersion]);

changes += modifyValue(opts.debugids, mConfigVersions[kIndexDebug]);

changes += modifyValue(opts.release, mConfigRelease);

changes += modifyValue(opts.useUnitTests, mConfigUnittest);

changes += modifyValue(opts.isX86\_64, mConfigX64);

changes += modifyValue(opts.useMSVCRT(), mConfigMSVCRT);

changes += modifyValue(opts.cov, mConfigCoverage);

changes += modifyValue(opts.doDocComments, mConfigDoc);

changes += modifyValue(opts.noboundscheck, mConfigNoBoundsCheck);

changes += modifyValue(opts.compiler, mConfigCompiler);

}

return changes != 0;

}

Config GetConfig()

{

if(!mConfig)

{

UpdateConfig();

}

return mConfig;

}

// ConfigModifiedListener

override void OnConfigModified()

{

OnConfigModified(false);

}

void OnConfigModified(bool force)

{

int changes = UpdateConfig();

changes += modifyValue(Package.GetGlobalOptions().ColorizeVersions, mColorizeVersions);

changes += modifyValue(Package.GetGlobalOptions().ColorizeCoverage, mColorizeCoverage);

if(changes || force)

{

mLineState[] = -1;

mSource.ReColorizeLines(0, -1);

}

}

//////////////////////////////////////////////////////////

static int[] ReadCoverageFile(string lstname, out float coveragePercent)

{

coveragePercent = -1;

try

{

char[] lst = cast(char[]) std.file.read(lstname);

char[][] lines = splitLines(lst);

int[] coverage = new int[lines.length];

foreach(i, ln; lines)

{

auto pos = std.string.indexOf(ln, '|');

int cov = -1;

if(pos > 0)

{

auto num = strip(ln[0..pos]);

if(num.length)

cov = parse!int(num);

}

coverage[i] = cov;

}

if (lines.length > 0)

{

char[] ln = lines[$-1];

auto pos = std.string.indexOf(ln, "% covered");

if(pos > 0)

{

auto end = pos;

while(pos > 0 && isDigit(ln[pos-1]) || ln[pos - 1] == '.')

pos--;

auto num = ln[pos..end];

if(num.length)

coveragePercent = parse!float(num); // very last entry is percent

}

}

return coverage;

}

catch(Error)

{

}

return null;

}

bool lastCoverageFileIsValid()

{

return (mLastCoverageFile.length > 0 && std.file.exists(mLastCoverageFile) && std.file.isFile(mLastCoverageFile));

}

bool FindCoverageFile()

{

if(lastCoverageFileIsValid())

return true;

mLastCoverageFile = Package.GetGlobalOptions().findCoverageFile(mSource.GetFileName());

return lastCoverageFileIsValid();

}

bool UpdateCoverage(bool force)

{

if(mColorizeCoverage)

{

auto now = Clock.currTime();

if(!force && mLastTestCoverageFile + dur!"seconds"(2) >= now)

return false;

mLastTestCoverageFile = now;

if(FindCoverageFile())

{

auto lsttm = std.file.timeLastModified(mLastCoverageFile);

auto srctm = std.file.timeLastModified(mSource.GetFileName());

if (lsttm < srctm)

{

ClearCoverage();

}

else if(force || lsttm != mLastModifiedCoverageFile)

{

mLastModifiedCoverageFile = lsttm;

mCoverage = ReadCoverageFile(mLastCoverageFile, mCoveragePercent);

mSource.ReColorizeLines(0, -1);

}

return true;

}

}

ClearCoverage();

return false;

}

void ClearCoverage()

{

mCoverage = mCoverage.init;

if(mLastModifiedCoverageFile != SysTime(0))

{

mLastCoverageFile = null;

mLastModifiedCoverageFile = SysTime(0);

mSource.ReColorizeLines(0, -1);

}

}

}

// This file is part of Visual D

//

// Visual D integrates the D programming language into Visual Studio

// Copyright (c) 2010 by Rainer Schuetze, All Rights Reserved

//

// Distributed under the Boost Software License, Version 1.0.

// See accompanying file LICENSE\_1\_0.txt or copy at <http://www.boost.org/LICENSE_1_0.txt>

module visuald.completion;

import visuald.windows;

import std.ascii;

import std.string;

import std.utf;

import std.file;

import std.path;

import std.algorithm;

import std.array;

import std.conv;

import stdext.array;

import stdext.file;

import stdext.ddocmacros;

import visuald.comutil;

import visuald.logutil;

import visuald.hierutil;

import visuald.fileutil;

import visuald.pkgutil;

import visuald.stringutil;

import visuald.dpackage;

import visuald.dproject;

import visuald.dlangsvc;

import visuald.dimagelist;

import visuald.dllmain;

import visuald.config;

import visuald.intellisense;

import vdc.lexer;

import sdk.port.vsi;

import sdk.win32.commctrl;

import sdk.vsi.textmgr;

import sdk.vsi.textmgr2;

import sdk.vsi.vsshell;

///////////////////////////////////////////////////////////////

const int kCompletionSearchLines = 5000;

class ImageList {};

struct Declaration

{

}

class Declarations

{

string[] mNames;

string[] mDescriptions;

int[] mGlyphs;

int mExpansionState = kStateInit;

enum

{

kStateInit,

kStateImport,

kStateSemantic,

kStateNearBy,

kStateSymbols,

kStateCaseInsensitive

}

this()

{

}

int GetCount()

{

return mNames.length;

}

dchar OnAutoComplete(IVsTextView textView, string committedWord, dchar commitChar, int commitIndex)

{

return 0;

}

bool splitName(int index, string\* pname, string\* ptext, string\* ptype, string\* pdesc)

{

if(index < 0 || index >= mNames.length)

return false;

string s = mNames[index];

auto pos1 = indexOf(s, ':');

string name, text, type, desc;

if(pos1 >= 0)

{

name = s[0 .. pos1];

auto pos2 = indexOf(s[pos1 + 1 .. $], ':');

if(pos2 >= 0)

{

type = s[pos1 + 1 .. pos1 + 1 + pos2];

desc = s[pos1 + 1 + pos2 + 1 .. $];

}

else

type = s[pos1 + 1 .. $];

}

else

name = s;

auto pos3 = indexOf(name, "|");

if(pos3 >= 0)

{

text = name[pos3 + 1 .. $];

name = name[0 .. pos3];

}

else

text = name;

if(pname)

\*pname = name;

if(ptext)

\*ptext = text;

if(ptype)

\*ptype = type;

if(pdesc)

\*pdesc = desc;

return true;

}

int GetGlyph(int index)

{

version(none)

{

if(index < 0 || index >= mGlyphs.length)

return 0;

return mGlyphs[index];

}

else

{

string type;

if(!splitName(index, null, null, &type, null))

return CSIMG\_DMODULE;

switch(type)

{

case "KW": return CSIMG\_KEYWORD;

case "PROP": return CSIMG\_PROPERTY;

case "TEXT": return 3;

case "MOD": return CSIMG\_DMODULE;

case "DIR": return CSIMG\_DFOLDER;

case "PKG": return CSIMG\_PACKAGE;

case "MTHD": return CSIMG\_MEMBER;

case "STRU": return CSIMG\_STRUCT;

case "UNIO": return CSIMG\_UNION;

case "CLSS": return CSIMG\_CLASS;

case "IFAC": return CSIMG\_INTERFACE;

case "TMPL": return CSIMG\_TEMPLATE;

case "ENUM": return CSIMG\_ENUM;

case "EVAL": return CSIMG\_ENUMMEMBER;

case "NMIX": return CSIMG\_UNKNOWN2;

case "VAR": return CSIMG\_FIELD;

case "OVR": return CSIMG\_UNKNOWN3;

case "ICN": return CSIMG\_UNKNOWN3;

default: return 0;

}

}

}

string GetDisplayText(int index)

{

return GetName(index);

}

string GetDescription(int index)

{

version(none)

{

if(index < 0 || index >= mDescriptions.length)

return "";

return mDescriptions[index];

}

else

{

string desc;

splitName(index, null, null, null, &desc);

desc = replace(desc, "\a", "\n");

string res = phobosDdocExpand(desc);

return res;

}

}

string GetName(int index)

{

string name;

splitName(index, &name, null, null, null);

return name;

}

string GetText(IVsTextView view, int index)

{

string text;

splitName(index, null, &text, null, null);

if(text.indexOf('\a') >= 0)

{

string nl = "\r\n";

if(view)

{

// copy indentation from current line

int line, idx;

if(view.GetCaretPos(&line, &idx) == S\_OK)

{

IVsTextLines pBuffer;

if(view.GetBuffer(&pBuffer) == S\_OK)

{

LINEDATA lineData;

if(pBuffer.GetLineData(line, &lineData, null) == S\_OK)

{

switch(lineData.iEolType)

{

default:

case eolCRLF: nl = "\r\n"; break;

case eolCR: nl = "\r"; break;

case eolLF: nl = "\n"; break;

}

pBuffer.ReleaseLineData(&lineData);

}

BSTR btext;

if(pBuffer.GetLineText(line, 0, line, idx, &btext) == S\_OK)

{

string txt = detachBSTR(btext);

size\_t p = 0;

while(p < txt.length && isWhite(txt[p]))

p++;

nl ~= txt[0 .. p];

}

release(pBuffer);

}

}

}

text = text.replace("\a", nl);

}

return text;

}

bool GetInitialExtent(IVsTextView textView, int\* line, int\* startIdx, int\* endIdx)

{

\*line = 0;

\*startIdx = \*endIdx = 0;

return false;

}

void GetBestMatch(string textSoFar, int\* index, bool \*uniqueMatch)

{

\*index = 0;

\*uniqueMatch = false;

}

bool IsCommitChar(string textSoFar, int index, dchar ch)

{

return ch == '\n' || ch == '\r'; // !(isAlphaNum(ch) || ch == '\_');

}

string OnCommit(IVsTextView textView, string textSoFar, dchar ch, int index, ref TextSpan initialExtent)

{

return GetText(textView, index); // textSoFar;

}

///////////////////////////////////////////////////////////////

bool ImportExpansions(string imp, string file)

{

string[] imports = GetImportPaths(file);

string dir;

int dpos = lastIndexOf(imp, '.');

if(dpos >= 0)

{

dir = replace(imp[0 .. dpos], ".", "\\");

imp = imp[dpos + 1 .. $];

}

int namesLength = mNames.length;

foreach(string impdir; imports)

{

impdir = impdir ~ dir;

if(!isExistingDir(impdir))

continue;

foreach(string name; dirEntries(impdir, SpanMode.shallow))

{

string base = baseName(name);

string ext = toLower(extension(name));

bool canImport = false;

bool issubdir = isDir(name);

if(issubdir)

canImport = (ext.length == 0);

else if(ext == ".d" || ext == ".di")

{

base = base[0 .. $-ext.length];

canImport = true;

}

if(canImport && base.startsWith(imp))

{

string txt = base ~ (issubdir ? ":DIR" : ":MOD");

addunique(mNames, txt);

}

}

}

return mNames.length > namesLength;

}

bool ImportExpansions(IVsTextView textView, Source src)

{

int line, idx;

if(int hr = textView.GetCaretPos(&line, &idx))

return false;

wstring wimp = src.GetImportModule(line, idx, true);

if(wimp.empty)

return false;

string txt = to!string(wimp);

ImportExpansions(txt, src.GetFileName());

return true;

}

///////////////////////////////////////////////////////////////

string GetTokenBeforeCaret(IVsTextView textView, Source src)

{

int line, idx;

int hr = textView.GetCaretPos(&line, &idx);

assert(hr == S\_OK);

int startIdx, endIdx;

if(!src.GetWordExtent(line, idx, WORDEXT\_FINDTOKEN, startIdx, endIdx))

return "";

wstring txt = src.GetText(line, startIdx, line, idx);

return toUTF8(txt);

}

bool NearbyExpansions(IVsTextView textView, Source src)

{

if(!Package.GetGlobalOptions().expandFromBuffer)

return false;

int line, idx;

if(int hr = textView.GetCaretPos(&line, &idx))

return false;

int lineCount;

src.mBuffer.GetLineCount(&lineCount);

//mNames.length = 0;

int start = max(0, line - kCompletionSearchLines);

int end = min(lineCount, line + kCompletionSearchLines);

string tok = GetTokenBeforeCaret(textView, src);

if(tok.length && !dLex.isIdentifierCharOrDigit(tok.front))

tok = "";

int iState = src.mColorizer.GetLineState(start);

if(iState == -1)

return false;

int namesLength = mNames.length;

for(int ln = start; ln < end; ln++)

{

wstring text = src.GetText(ln, 0, ln, -1);

uint pos = 0;

while(pos < text.length)

{

uint ppos = pos;

int type = dLex.scan(iState, text, pos);

if(ln != line || pos < idx || ppos > idx)

if(type == TokenCat.Identifier || type == TokenCat.Keyword)

{

string txt = toUTF8(text[ppos .. pos]);

if(txt.startsWith(tok))

addunique(mNames, txt);

}

}

}

return mNames.length > namesLength;

}

////////////////////////////////////////////////////////////////////////

bool SymbolExpansions(IVsTextView textView, Source src)

{

if(!Package.GetGlobalOptions().expandFromJSON)

return false;

string tok = GetTokenBeforeCaret(textView, src);

if(tok.length && !dLex.isIdentifierCharOrDigit(tok.front))

tok = "";

if(!tok.length)

return false;

int namesLength = mNames.length;

addunique(mNames, Package.GetLibInfos().findCompletions(tok, true));

return mNames.length > namesLength;

}

///////////////////////////////////////////////////////////////////////////

bool SemanticExpansions(IVsTextView textView, Source src)

{

if(!Package.GetGlobalOptions().expandFromSemantics)

return false;

try

{

string tok = GetTokenBeforeCaret(textView, src);

if(tok.length && !dLex.isIdentifierCharOrDigit(tok.front))

tok = "";

int line, idx;

int hr = textView.GetCaretPos(&line, &idx);

src.ensureCurrentTextParsed(); // pass new text before expansion request

auto langsvc = Package.GetLanguageService();

mPendingSource = src;

mPendingView = textView;

mPendingRequest = langsvc.GetSemanticExpansions(src, tok, line, idx, &OnExpansions);

return true;

}

catch(Error e)

{

writeToBuildOutputPane(e.msg);

}

return false;

}

extern(D) void OnExpansions(uint request, string filename, string tok, int line, int idx, string[] symbols)

{

if(request != mPendingRequest)

return;

if(symbols.length > 0 && mPendingSource)

{

// split after second ':' to combine same name and type

static string splitName(string name, ref string desc)

{

auto pos = name.indexOf(':');

if(pos < 0)

return name;

pos = name.indexOf(':', pos + 1);

if(pos < 0)

return name;

desc = name[pos..$];

return name[0..pos];

}

// go through assoc array for faster uniqueness check

string[string] names;

foreach(n; mNames)

{

string desc;

string name = splitName(n, desc);

names[name] = desc;

}

foreach(s; symbols)

{

string desc;

string name = splitName(s, desc);

if(auto p = name in names)

\*p ~= "\a\a" ~ desc[1..$]; // strip ":"

else

names[name] = desc;

}

mNames.length = names.length;

size\_t i = 0;

foreach(n, desc; names)

mNames[i++] = n ~ desc;

sort!("icmp(a, b) < 0", SwapStrategy.stable)(mNames);

mPendingSource.GetCompletionSet().Init(mPendingView, this, false);

}

mPendingRequest = 0;

mPendingView = null;

mPendingSource = null;

}

uint mPendingRequest;

IVsTextView mPendingView;

Source mPendingSource;

////////////////////////////////////////////////////////////////////////

bool StartExpansions(IVsTextView textView, Source src, bool autoInsert)

{

mNames = mNames.init;

mExpansionState = kStateInit;

if(!\_MoreExpansions(textView, src))

return false;

if(autoInsert)

{

while(GetCount() == 1 && \_MoreExpansions(textView, src)) {}

if(GetCount() == 1)

{

int line, idx, startIdx, endIdx;

textView.GetCaretPos(&line, &idx);

if(src.GetWordExtent(line, idx, WORDEXT\_FINDTOKEN, startIdx, endIdx))

{

wstring txt = to!wstring(GetText(textView, 0));

TextSpan changedSpan;

src.mBuffer.ReplaceLines(line, startIdx, line, endIdx, txt.ptr, txt.length, &changedSpan);

return true;

}

}

}

src.GetCompletionSet().Init(textView, this, false);

return true;

}

bool \_MoreExpansions(IVsTextView textView, Source src)

{

switch(mExpansionState)

{

case kStateInit:

if(ImportExpansions(textView, src))

{

mExpansionState = kStateSymbols; // do not try other symbols but file imports

return true;

}

goto case;

case kStateImport:

if(SemanticExpansions(textView, src))

{

mExpansionState = kStateSemantic;

return true;

}

goto case;

case kStateSemantic:

if(NearbyExpansions(textView, src))

{

mExpansionState = kStateNearBy;

return true;

}

goto case;

case kStateNearBy:

if(SymbolExpansions(textView, src))

{

mExpansionState = kStateSymbols;

return true;

}

goto default;

default:

break;

}

return false;

}

bool MoreExpansions(IVsTextView textView, Source src)

{

\_MoreExpansions(textView, src);

src.GetCompletionSet().Init(textView, this, false);

return true;

}

void StopExpansions()

{

mPendingRequest = 0;

mPendingView = null;

mPendingSource = null;

}

}

class CompletionSet : DisposingComObject, IVsCompletionSet, IVsCompletionSetEx

{

HIMAGELIST mImageList;

bool mDisplayed;

bool mCompleteWord;

string mCommittedWord;

dchar mCommitChar;

int mCommitIndex;

IVsTextView mTextView;

Declarations mDecls;

Source mSource;

TextSpan mInitialExtent;

bool mIsCommitted;

bool mWasUnique;

this(ImageList imageList, Source source)

{

mImageList = LoadImageList(g\_hInst, MAKEINTRESOURCEA(BMP\_COMPLETION), 16, 16);

mSource = source;

}

override HRESULT QueryInterface(in IID\* riid, void\*\* pvObject)

{

if(queryInterface!(IVsCompletionSetEx) (this, riid, pvObject))

return S\_OK;

if(queryInterface!(IVsCompletionSet) (this, riid, pvObject))

return S\_OK;

if(\*riid == uuid\_IVsCoTaskMemFreeMyStrings) // avoid log message, implement?

return E\_NOTIMPL;

return super.QueryInterface(riid, pvObject);

}

void Init(IVsTextView textView, Declarations declarations, bool completeWord)

{

Close();

mTextView = textView;

mDecls = declarations;

mCompleteWord = completeWord;

//check if we have members

int count = mDecls.GetCount();

if (count <= 0) return;

//initialise and refresh

UpdateCompletionFlags flags = UCS\_NAMESCHANGED;

if (mCompleteWord)

flags |= UCS\_COMPLETEWORD;

mWasUnique = false;

int hr = textView.UpdateCompletionStatus(this, flags);

assert(hr == S\_OK);

mDisplayed = (!mWasUnique || !completeWord);

}

override void Dispose()

{

Close();

//if (imageList != null) imageList.Dispose();

if(mImageList)

{

ImageList\_Destroy(mImageList);

mImageList = null;

}

}

void Close()

{

if (mDisplayed && mTextView)

{

// Here we can't throw or exit because we need to call Dispose on

// the disposable membres.

try {

mTextView.UpdateCompletionStatus(null, 0);

} catch (Exception e) {

}

}

mDisplayed = false;

mTextView = null;

mDecls = null;

}

dchar OnAutoComplete()

{

mIsCommitted = false;

if (mDecls)

return mDecls.OnAutoComplete(mTextView, mCommittedWord, mCommitChar, mCommitIndex);

return '\0';

}

//--------------------------------------------------------------------------

//IVsCompletionSet methods

//--------------------------------------------------------------------------

override int GetImageList(HANDLE \*phImages)

{

mixin(LogCallMix);

\*phImages = cast(HANDLE)mImageList;

return S\_OK;

}

override int GetFlags()

{

mixin(LogCallMix);

return CSF\_HAVEDESCRIPTIONS | CSF\_CUSTOMCOMMIT | CSF\_INITIALEXTENTKNOWN | CSF\_CUSTOMMATCHING;

}

override int GetCount()

{

mixin(LogCallMix);

return mDecls.GetCount();

}

override int GetDisplayText(in int index, WCHAR\*\* text, int\* glyph)

{

//mixin(LogCallMix);

if (glyph)

\*glyph = mDecls.GetGlyph(index);

\*text = allocBSTR(mDecls.GetDisplayText(index));

return S\_OK;

}

override int GetDescriptionText(in int index, BSTR\* description)

{

mixin(LogCallMix2);

\*description = allocBSTR(mDecls.GetDescription(index));

return S\_OK;

}

override int GetInitialExtent(int\* line, int\* startIdx, int\* endIdx)

{

mixin(LogCallMix);

int idx;

int hr = S\_OK;

if (mDecls.GetInitialExtent(mTextView, line, startIdx, endIdx))

goto done;

hr = mTextView.GetCaretPos(line, &idx);

assert(hr == S\_OK);

hr = GetTokenExtent(\*line, idx, \*startIdx, \*endIdx);

done:

// Remember the initial extent so we can pass it along on the commit.

mInitialExtent.iStartLine = mInitialExtent.iEndLine = \*line;

mInitialExtent.iStartIndex = \*startIdx;

mInitialExtent.iEndIndex = \*endIdx;

//assert(TextSpanHelper.ValidCoord(mSource, line, startIdx) &&

// TextSpanHelper.ValidCoord(mSource, line, endIdx));

return hr;

}

int GetTokenExtent(int line, int idx, out int startIdx, out int endIdx)

{

int hr = S\_OK;

bool rc = mSource.GetWordExtent(line, idx, WORDEXT\_FINDTOKEN, startIdx, endIdx);

if (!rc && idx > 0)

{

//rc = mSource.GetWordExtent(line, idx - 1, WORDEXT\_FINDTOKEN, startIdx, endIdx);

if (!rc)

{

// Must stop core text editor from looking at startIdx and endIdx since they are likely

// invalid. So we must return a real failure here, not just S\_FALSE.

startIdx = endIdx = idx;

hr = E\_NOTIMPL;

}

}

// make sure the span is positive.

endIdx = max(endIdx, idx);

return hr;

}

override int GetBestMatch(in WCHAR\* wtextSoFar, in int length, int\* index, uint\* flags)

{

mixin(LogCallMix);

\*flags = 0;

\*index = 0;

bool uniqueMatch = false;

string textSoFar = to\_string(wtextSoFar);

if (textSoFar.length != 0)

{

mDecls.GetBestMatch(textSoFar, index, &uniqueMatch);

if (\*index < 0 || \*index >= mDecls.GetCount())

{

\*index = 0;

uniqueMatch = false;

} else {

// Indicate that we want to select something in the list.

\*flags = GBM\_SELECT;

}

}

else if (mDecls.GetCount() == 1 && mCompleteWord)

{

// Only one entry, and user has invoked "word completion", then

// simply select this item.

\*index = 0;

\*flags = GBM\_SELECT;

uniqueMatch = true;

}

if (uniqueMatch)

{

\*flags |= GBM\_UNIQUE;

mWasUnique = true;

}

return S\_OK;

}

override int OnCommit(in WCHAR\* wtextSoFar, in int index, in BOOL selected, in WCHAR commitChar, BSTR\* completeWord)

{

mixin(LogCallMix);

dchar ch = commitChar;

bool isCommitChar = true;

string textSoFar = to\_string(wtextSoFar);

if (commitChar != 0)

{

// if the char is in the list of given member names then obviously it

// is not a commit char.

int i = textSoFar.length;

for (int j = 0, n = mDecls.GetCount(); j < n; j++)

{

string name = mDecls.GetText(mTextView, j);

if (name.length > i && name[i] == commitChar)

{

if (i == 0 || name[0 .. i] == textSoFar)

goto nocommit; // cannot be a commit char if it is an expected char in a matching name

}

}

isCommitChar = mDecls.IsCommitChar(textSoFar, (selected == 0) ? -1 : index, ch);

}

if (isCommitChar)

{

mCommittedWord = mDecls.OnCommit(mTextView, textSoFar, ch, selected == 0 ? -1 : index, mInitialExtent);

\*completeWord = allocBSTR(mCommittedWord);

mCommitChar = ch;

mCommitIndex = index;

mIsCommitted = true;

return S\_OK;

}

nocommit:

// S\_FALSE return means the character is not a commit character.

\*completeWord = allocBSTR(textSoFar);

return S\_FALSE;

}

override int Dismiss()

{

mixin(LogCallMix);

mDisplayed = false;

return S\_OK;

}

// IVsCompletionSetEx Members

override int CompareItems(in BSTR bstrSoFar, in BSTR bstrOther, in int lCharactersToCompare, int\* plResult)

{

mixin(LogCallMix);

\*plResult = 0;

return E\_NOTIMPL;

}

override int IncreaseFilterLevel(in int iSelectedItem)

{

mixin(LogCallMix2);

return E\_NOTIMPL;

}

override int DecreaseFilterLevel(in int iSelectedItem)

{

mixin(LogCallMix2);

return E\_NOTIMPL;

}

override int GetCompletionItemColor(in int iIndex, COLORREF\* dwFGColor, COLORREF\* dwBGColor)

{

mixin(LogCallMix);

\*dwFGColor = \*dwBGColor = 0;

return E\_NOTIMPL;

}

override int GetFilterLevel(int\* iFilterLevel)

{

mixin(LogCallMix2);

\*iFilterLevel = 0;

return E\_NOTIMPL;

}

override int OnCommitComplete()

{

mixin(LogCallMix);

/+

if(CodeWindowManager mgr = mSource.LanguageService.GetCodeWindowManagerForView(mTextView))

if (ViewFilter filter = mgr.GetFilter(mTextView))

filter.OnAutoComplete();

+/

return S\_OK;

}

}

//-------------------------------------------------------------------------------------

class MethodData : DisposingComObject, IVsMethodData

{

IServiceProvider mProvider;

IVsMethodTipWindow mMethodTipWindow;

Definition[] mMethods;

bool mTypePrefixed = true;

int mCurrentParameter;

int mCurrentMethod;

bool mDisplayed;

IVsTextView mTextView;

TextSpan mContext;

this()

{

auto uuid = uuid\_coclass\_VsMethodTipWindow;

mMethodTipWindow = VsLocalCreateInstance!IVsMethodTipWindow (&uuid, sdk.win32.wtypes.CLSCTX\_INPROC\_SERVER);

if (mMethodTipWindow)

mMethodTipWindow.SetMethodData(this);

}

override HRESULT QueryInterface(in IID\* riid, void\*\* pvObject)

{

if(queryInterface!(IVsMethodData) (this, riid, pvObject))

return S\_OK;

return super.QueryInterface(riid, pvObject);

}

void Refresh(IVsTextView textView, Definition[] methods, int currentParameter, TextSpan context)

{

if (!mDisplayed)

mCurrentMethod = 0; // methods.DefaultMethod;

mContext = context;

mMethods = mMethods.init;

defLoop:

foreach(ref def; methods)

{

foreach(ref d; mMethods)

if(d.type == def.type)

{

if (!d.inScope.endsWith(" ..."))

d.inScope ~= " ...";

continue defLoop;

}

mMethods ~= def;

}

// Apparently this Refresh() method is called as a result of event notification

// after the currentMethod is changed, so we do not want to Dismiss anything or

// reset the currentMethod here.

//Dismiss();

mTextView = textView;

mCurrentParameter = currentParameter;

AdjustCurrentParameter(0);

}

void AdjustCurrentParameter(int increment)

{

mCurrentParameter += increment;

if (mCurrentParameter < 0)

mCurrentParameter = -1;

else if (mCurrentParameter >= GetParameterCount(mCurrentMethod))

mCurrentParameter = GetParameterCount(mCurrentMethod);

UpdateView();

}

void Close()

{

Dismiss();

mTextView = null;

mMethods = null;

}

void Dismiss()

{

if (mDisplayed && mTextView)

mTextView.UpdateTipWindow(mMethodTipWindow, UTW\_DISMISS);

OnDismiss();

}

override void Dispose()

{

Close();

if (mMethodTipWindow)

mMethodTipWindow.SetMethodData(null);

mMethodTipWindow = release(mMethodTipWindow);

}

//IVsMethodData

override int GetOverloadCount()

{

if (!mTextView || mMethods.length == 0)

return 0;

return mMethods.length;

}

override int GetCurMethod()

{

return mCurrentMethod;

}

override int NextMethod()

{

if (mCurrentMethod < GetOverloadCount() - 1)

mCurrentMethod++;

return mCurrentMethod;

}

override int PrevMethod()

{

if (mCurrentMethod > 0)

mCurrentMethod--;

return mCurrentMethod;

}

override int GetParameterCount(in int method)

{

if (mMethods.length == 0)

return 0;

if (method < 0 || method >= GetOverloadCount())

return 0;

return mMethods[method].GetParameterCount();

}

override int GetCurrentParameter(in int method)

{

return mCurrentParameter;

}

override void OnDismiss()

{

mTextView = null;

mMethods = mMethods.init;

mCurrentMethod = 0;

mCurrentParameter = 0;

mDisplayed = false;

}

override void UpdateView()

{

if (mTextView && mMethodTipWindow)

{

mTextView.UpdateTipWindow(mMethodTipWindow, UTW\_CONTENTCHANGED | UTW\_CONTEXTCHANGED);

mDisplayed = true;

}

}

override int GetContextStream(int\* pos, int\* length)

{

\*pos = 0;

\*length = 0;

int line, idx, vspace, endpos;

if(HRESULT rc = mTextView.GetCaretPos(&line, &idx))

return rc;

line = max(line, mContext.iStartLine);

if(HRESULT rc = mTextView.GetNearestPosition(line, mContext.iStartIndex, pos, &vspace))

return rc;

line = max(line, mContext.iEndLine);

if(HRESULT rc = mTextView.GetNearestPosition(line, mContext.iEndIndex, &endpos, &vspace))

return rc;

\*length = endpos - \*pos;

return S\_OK;

}

override WCHAR\* GetMethodText(in int method, in MethodTextType type)

{

if (mMethods.length == 0)

return null;

if (method < 0 || method >= GetOverloadCount())

return null;

string result;

//a type

if ((type == MTT\_TYPEPREFIX && mTypePrefixed) ||

(type == MTT\_TYPEPOSTFIX && !mTypePrefixed))

{

string str = mMethods[method].GetReturnType();

if (str.length == 0)

return null;

result = str; // mMethods.TypePrefix + str + mMethods.TypePostfix;

}

else

{

//other

switch (type) {

case MTT\_OPENBRACKET:

result = "("; // mMethods.OpenBracket;

break;

case MTT\_CLOSEBRACKET:

result = ")"; // mMethods.CloseBracket;

break;

case MTT\_DELIMITER:

result = ","; // mMethods.Delimiter;

break;

case MTT\_NAME:

result = mMethods[method].name;

break;

case MTT\_DESCRIPTION:

if(mMethods[method].help.length)

result = phobosDdocExpand(mMethods[method].help);

else if(mMethods[method].line > 0)

result = format("%s %s @ %s(%d)", mMethods[method].kind, mMethods[method].inScope, mMethods[method].filename, mMethods[method].line);

break;

case MTT\_TYPEPREFIX:

case MTT\_TYPEPOSTFIX:

default:

break;

}

}

return result.length == 0 ? null : allocBSTR(result); // produces leaks?

}

override WCHAR\* GetParameterText(in int method, in int parameter, in ParameterTextType type)

{

if (mMethods.length == 0)

return null;

if (method < 0 || method >= GetOverloadCount())

return null;

if (parameter < 0 || parameter >= GetParameterCount(method))

return null;

string name;

string description;

string display;

mMethods[method].GetParameterInfo(parameter, name, display, description);

string result;

switch (type) {

case PTT\_NAME:

result = name;

break;

case PTT\_DESCRIPTION:

result = description;

break;

case PTT\_DECLARATION:

result = display;

break;

default:

break;

}

return result.length == 0 ? null : allocBSTR(result); // produces leaks?

}

}

// This file is part of Visual D

//

// Visual D integrates the D programming language into Visual Studio

// Copyright (c) 2010 by Rainer Schuetze, All Rights Reserved

//

// Distributed under the Boost Software License, Version 1.0.

// See accompanying file LICENSE\_1\_0.txt or copy at <http://www.boost.org/LICENSE_1_0.txt>

module visuald.comutil;

import visuald.windows;

import core.stdc.string;

import core.stdc.stdlib;

import std.string;

import std.utf;

import std.traits;

//import variant;

public import sdk.port.base;

public import sdk.port.stdole2;

public import stdext.com;

import sdk.win32.oleauto;

import sdk.win32.objbase;

//debug debug = COM;

//debug(COM) debug = COM\_DTOR; // causes crashes because logCall needs GC, but finalizer called from within GC

//debug(COM) debug = COM\_ADDREL;

import core.runtime;

//debug(COM\_ADDREL) debug static import rsgc.gc;

import core.memory;

debug import core.stdc.stdio;

import visuald.logutil;

extern (C) void \_d\_callfinalizer(void \*p);

///////////////////////////////////////////////////////////////////////////////

///////////////////////////////////////////////////////////////////////////////

uint Advise(Interface)(IUnknown pSource, IUnknown pSink)

{

auto container = ComPtr!(IConnectionPointContainer)(pSource);

if(container)

{

ComPtr!(IConnectionPoint) point;

if(container.FindConnectionPoint(&Interface.iid, &point.ptr) == S\_OK)

{

uint cookie;

if(point.Advise(pSink, &cookie) == S\_OK)

return cookie;

}

}

return 0;

}

uint Unadvise(Interface)(IUnknown pSource, uint cookie)

{

auto container = ComPtr!(IConnectionPointContainer)(pSource);

if(container)

{

ComPtr!(IConnectionPoint) point;

if(container.FindConnectionPoint(&Interface.iid, &point.ptr) == S\_OK)

{

if(point.Unadvise(cookie) == S\_OK)

return cookie;

}

}

return 0;

}

///////////////////////////////////////////////////////////////////////////////

class DComObject : ComObject

{

debug

{

\_\_gshared LONG sCountCreated;

\_\_gshared LONG sCountInstances;

\_\_gshared LONG sCountReferenced;

debug(COM\_ADDREL) \_\_gshared int[LONG] sReferencedObjects;

enum size\_t WEAK\_PTR\_XOR = 0x80000000;

alias AssociativeArray!(LONG, int) \_wa1; // fully instantiate type info

}

debug

{

this()

{

void\* vthis = cast(void\*) this;

debug(COM) logCall("ctor %s this = %s", this, vthis);

debug(COM\_ADDREL) synchronized(DComObject.classinfo) sReferencedObjects[cast(size\_t)vthis^WEAK\_PTR\_XOR] = 0;

InterlockedIncrement(&sCountInstances);

InterlockedIncrement(&sCountCreated);

}

~this()

{

// logCall needs GC, but finalizer called from within GC

void\* vthis = cast(void\*) this;

debug(COM\_DTOR) logCall("dtor %s this = %s", this, vthis);

debug(COM\_ADDREL)

synchronized(DComObject.classinfo)

if(auto p = (cast(size\_t)vthis^WEAK\_PTR\_XOR) in sReferencedObjects)

\*p = -1;

InterlockedDecrement(&sCountInstances);

}

static void showCOMleaks()

{

alias OutputDebugStringA ods;

char[1024] sbuf;

sprintf(sbuf.ptr, "%d COM objects created\n", sCountCreated); ods(sbuf.ptr);

sprintf(sbuf.ptr, "%d COM objects never destroyed (no final collection run yet!)\n", sCountInstances); ods(sbuf.ptr);

sprintf(sbuf.ptr, "%d COM objects not fully dereferenced\n", sCountReferenced); ods(sbuf.ptr);

debug(COM\_ADDREL)

foreach(p, b; sReferencedObjects)

{

void\* q = cast(void\*)(p^WEAK\_PTR\_XOR);

if(b > 0)

{

sprintf(sbuf.ptr, " leaked COM object: %p %s\n", q, (cast(Object)q).classinfo.name.ptr); ods(sbuf.ptr);

}

else if(b == 0)

{

sprintf(sbuf.ptr, " not collected: %p %s\n", q, (cast(Object)q).classinfo.name.ptr); ods(sbuf.ptr);

}

version(none)

if(b >= 0)

{

auto r = rsgc.gc.gc\_findReference(q, (cast(Object)q).classinfo.init.length);

auto base = rsgc.gc.gc\_addrOf(r);

string type = "unknown";

if(base)

{

int attr = rsgc.gc.gc\_getAttr(base);

if(attr & 1)

type = (cast(Object)base).classinfo.name;

}

sprintf(sbuf.ptr, " referenced by %p inside %p %s\n", r, base, type.ptr); ods(sbuf.ptr);

}

}

}

}

extern (System):

override HRESULT QueryInterface(in IID\* riid, void\*\* ppv)

{

HRESULT hr = super.QueryInterface(riid, ppv);

if (hr != S\_OK)

logCall("%s.QueryInterface(this=%s,riid=%s) no interface!", this, cast(void\*)this, \_toLog(riid));

return hr;

}

version(none) // copy for debugging

{

override ULONG AddRef()

{

return super.AddRef();

}

override ULONG Release()

{

return super.Release();

}

}

override ULONG AddRef()

{

LONG lRef = super.AddRef();

debug(COM\_ADDREL) logCall("addref %s this = %s ref = %d", this, cast(void\*)this, lRef);

if(lRef == 1)

{

debug InterlockedIncrement(&sCountReferenced);

//uint sz = this.classinfo.init.length;

debug void\* vthis = cast(void\*) this;

debug(COM) logCall("addroot %s this = %s", this, vthis);

debug(COM\_ADDREL)

synchronized(DComObject.classinfo) sReferencedObjects[cast(size\_t)vthis^WEAK\_PTR\_XOR] = 1;

}

return lRef;

}

override ULONG Release()

{

ULONG lRef = super.Release();

debug(COM\_ADDREL) logCall("release %s this = %s ref = %d", this, cast(void\*)this, lRef);

if (lRef == 0)

{

debug void\* vthis = cast(void\*) this;

debug(COM) logCall("delroot %s this = %s", this, vthis);

debug InterlockedDecrement(&sCountReferenced);

debug(COM\_ADDREL)

synchronized(DComObject.classinfo) sReferencedObjects[cast(size\_t)vthis^WEAK\_PTR\_XOR] = 0;

}

return lRef;

}

}

class DisposingComObject : DComObject

{

override ULONG Release()

{

assert(count > 0);

if(count == 1)

{

// avoid recursive delete if the object is temporarily ref-counted

// while executing Dispose()

count = 0x12345678;

Dispose();

assert(count == 0x12345678);

count = 1;

}

return super.Release();

}

abstract void Dispose();

}

/+

struct PARAMDATA

{

OLECHAR\* szName;

VARTYPE vtReturn;

}

struct METHODDATA

{

OLECHAR\* zName;

PARAMDATA\* ppData;

DISPID dispid;

uint iMeth;

CALLCONV cc;

uint cArgs;

ushort wFlags;

VARTYPE vtReturn;

}

struct INTERFACEDATA

{

METHODDATA\* pmethdata; // Pointer to an array of METHODDATAs.

uint cMembers; // Count of

}

+/

class DisposingDispatchObject : DisposingComObject, IDispatch

{

override HRESULT QueryInterface(in IID\* riid, void\*\* pvObject)

{

if(queryInterface!(IDispatch) (this, riid, pvObject))

return S\_OK;

return super.QueryInterface(riid, pvObject);

}

// IDispatch

override int GetTypeInfoCount(

/\* [out] \*/ UINT \*pctinfo)

{

//                mixin(LogCallMix);

\*pctinfo = 1;

return S\_OK;

}

override int GetTypeInfo(

/\* [in] \*/ in UINT iTInfo,

/\* [in] \*/ in LCID lcid,

/\* [out] \*/ ITypeInfo \*ppTInfo)

{

mixin(LogCallMix);

if(iTInfo != 0)

return returnError(E\_INVALIDARG);

\*ppTInfo = addref(getTypeHolder());

return S\_OK;

}

override int GetIDsOfNames(

/\* [in] \*/ in IID\* riid,

/\* [size\_is][in] \*/ in LPOLESTR \*rgszNames,

/\* [range][in] \*/ in UINT cNames,

/\* [in] \*/ in LCID lcid,

/\* [size\_is][out] \*/ DISPID \*rgDispId)

{

mixin(LogCallMix);

return getTypeHolder().GetIDsOfNames(rgszNames, cNames, rgDispId);

}

override int Invoke(

/\* [in] \*/ in DISPID dispIdMember,

/\* [in] \*/ in IID\* riid,

/\* [in] \*/ in LCID lcid,

/\* [in] \*/ in WORD wFlags,

/\* [out][in] \*/ DISPPARAMS \*pDispParams,

/\* [out] \*/ VARIANT \*pVarResult,

/\* [out] \*/ EXCEPINFO \*pExcepInfo,

/\* [out] \*/ UINT \*puArgErr)

{

mixin(LogCallMix);

return returnError(E\_NOTIMPL);

}

abstract ComTypeInfoHolder getTypeHolder();

}

struct DispatchData

{

int id;

string name;

FUNCDESC\* desc;

}

class ComTypeInfoHolder : DComObject, ITypeInfo

{

string[int] m\_pMap;

this()

{

}

override HRESULT QueryInterface(in IID\* riid, void\*\* pvObject)

{

if(queryInterface!(ITypeInfo) (this, riid, pvObject))

return S\_OK;

return super.QueryInterface(riid, pvObject);

}

override int GetTypeAttr(

/\* [out] \*/ TYPEATTR \*\*ppTypeAttr)

{

mixin(LogCallMix);

return returnError(E\_NOTIMPL);

}

override int GetTypeComp(

/\* [out] \*/ ITypeComp\* ppTComp)

{

mixin(LogCallMix);

return returnError(E\_NOTIMPL);

}

override int GetFuncDesc(

/\* [in] \*/ in UINT index,

/\* [out] \*/ FUNCDESC \*\*ppFuncDesc)

{

mixin(LogCallMix);

return returnError(E\_NOTIMPL);

}

override int GetVarDesc(

/\* [in] \*/ in UINT index,

/\* [out] \*/ VARDESC \*\*ppVarDesc)

{

mixin(LogCallMix);

return returnError(E\_NOTIMPL);

}

override int GetNames(

/\* [in] \*/ in MEMBERID memid,

/\* [length\_is][size\_is][out] \*/ BSTR \*rgBstrNames,

/\* [in] \*/ in UINT cMaxNames,

/\* [out] \*/ UINT \*pcNames)

{

mixin(LogCallMix);

return returnError(E\_NOTIMPL);

}

override int GetRefTypeOfImplType(

/\* [in] \*/ in UINT index,

/\* [out] \*/ HREFTYPE \*pRefType)

{

mixin(LogCallMix);

return returnError(E\_NOTIMPL);

}

override int GetImplTypeFlags(

/\* [in] \*/ in UINT index,

/\* [out] \*/ INT \*pImplTypeFlags)

{

mixin(LogCallMix);

return returnError(E\_NOTIMPL);

}

override int GetIDsOfNames(

/\* [size\_is][in] \*/ in LPOLESTR \*rgszNames,

/\* [in] \*/ in UINT cNames,

/\* [size\_is][out] \*/ MEMBERID \*pMemId)

{

mixin(LogCallMix);

return returnError(E\_NOTIMPL);

}

override int Invoke(

/\* [in] \*/ in PVOID pvInstance,

/\* [in] \*/ in MEMBERID memid,

/\* [in] \*/ in WORD wFlags,

/\* [out][in] \*/ DISPPARAMS \*pDispParams,

/\* [out] \*/ VARIANT \*pVarResult,

/\* [out] \*/ EXCEPINFO \*pExcepInfo,

/\* [out] \*/ UINT \*puArgErr)

{

mixin(LogCallMix2);

return returnError(E\_NOTIMPL);

}

override int GetDocumentation(

/\* [in] \*/ in MEMBERID memid,

/\* [out] \*/ BSTR \*pBstrName,

/\* [out] \*/ BSTR \*pBstrDocString,

/\* [out] \*/ DWORD \*pdwHelpContext,

/\* [out] \*/ BSTR \*pBstrHelpFile)

{

mixin(LogCallMix);

return returnError(E\_NOTIMPL);

}

override int GetDllEntry(

/\* [in] \*/ in MEMBERID memid,

/\* [in] \*/ in INVOKEKIND invKind,

/\* [out] \*/ BSTR \*pBstrDllName,

/\* [out] \*/ BSTR \*pBstrName,

/\* [out] \*/ WORD \*pwOrdinal)

{

mixin(LogCallMix);

return returnError(E\_NOTIMPL);

}

override int GetRefTypeInfo(

/\* [in] \*/ in HREFTYPE hRefType,

/\* [out] \*/ ITypeInfo\* ppTInfo)

{

mixin(LogCallMix);

return returnError(E\_NOTIMPL);

}

override int AddressOfMember(

/\* [in] \*/ in MEMBERID memid,

/\* [in] \*/ in INVOKEKIND invKind,

/\* [out] \*/ PVOID \*ppv)

{

mixin(LogCallMix);

return returnError(E\_NOTIMPL);

}

override int CreateInstance(

/\* [in] \*/ IUnknown pUnkOuter,

/\* [in] \*/ in IID\* riid,

/\* [iid\_is][out] \*/ PVOID \*ppvObj)

{

mixin(LogCallMix);

return returnError(E\_NOTIMPL);

}

override int GetMops(

/\* [in] \*/ in MEMBERID memid,

/\* [out] \*/ BSTR \*pBstrMops)

{

mixin(LogCallMix2);

return returnError(E\_NOTIMPL);

}

override int GetContainingTypeLib(

/\* [out] \*/ ITypeLib \*ppTLib,

/\* [out] \*/ UINT \*pIndex)

{

mixin(LogCallMix2);

return returnError(E\_NOTIMPL);

}

/\* [local] \*/ void ReleaseTypeAttr(

/\* [in] \*/ in TYPEATTR \*pTypeAttr)

{

mixin(LogCallMix);

//return returnError(E\_NOTIMPL);

}

/\* [local] \*/ void ReleaseFuncDesc(

/\* [in] \*/ in FUNCDESC \*pFuncDesc)

{

mixin(LogCallMix);

//return returnError(E\_NOTIMPL);

}

/\* [local] \*/ void ReleaseVarDesc(

/\* [in] \*/ in VARDESC \*pVarDesc)

{

mixin(LogCallMix);

//return returnError(E\_NOTIMPL);

}

}

// This file is part of Visual D

//

// Visual D integrates the D programming language into Visual Studio

// Copyright (c) 2010 by Rainer Schuetze, All Rights Reserved

//

// Distributed under the Boost Software License, Version 1.0.

// See accompanying file LICENSE\_1\_0.txt or copy at <http://www.boost.org/LICENSE_1_0.txt>

module visuald.config;

import std.string;

import std.conv;

import std.file;

import std.path;

import std.utf;

import std.array;

import std.exception;

import stdext.path;

import stdext.array;

import stdext.file;

import stdext.string;

import stdext.util;

import xml = visuald.xmlwrap;

import visuald.windows;

import sdk.port.vsi;

import sdk.win32.objbase;

import sdk.win32.oleauto;

import sdk.vsi.vsshell;

import sdk.vsi.vsshell80;

import sdk.vsi.vsshell110; // for IVsProfilableProjectCfg, etc

import visuald.comutil;

import visuald.logutil;

import visuald.hierutil;

import visuald.hierarchy;

import visuald.chiernode;

import visuald.dproject;

import visuald.dpackage;

import visuald.build;

import visuald.propertypage;

import visuald.stringutil;

import visuald.fileutil;

import visuald.lexutil;

import visuald.pkgutil;

import visuald.vdextensions;

version = hasOutputGroup;

// implementation of IVsProfilableProjectCfg is incomplete (profiler doesn't stop)

// but just providing proper output and debug information works for profiling as an executable

// version = hasProfilableConfig;

///////////////////////////////////////////////////////////////

const string[] kPlatforms = [ "Win32", "x64" ];

enum string kToolResourceCompiler = "Resource Compiler";

enum string kToolCpp = "C/C++";

const string kCmdLogFileExtension = "build";

version(hasProfilableConfig)

const GUID g\_unmarshalTargetInfoCLSID = uuid("002a2de9-8bb6-484d-980f-7e4ad4084715");

///////////////////////////////////////////////////////////////

T clone(T)(T object)

{

auto size = object.classinfo.init.length;

object = cast(T) ((cast(void\*)object) [0..size].dup.ptr );

//        object.\_\_monitor = null;

return object;

}

///////////////////////////////////////////////////////////////

ubyte toUbyte(string s) { return to!(ubyte)(s); }

float toFloat(string s) { return to!(float)(s); }

string uintToString(uint x) { return to!(string)(x); }

string toElem(bool b) { return b ? "1" : "0"; }

string toElem(float f) { return to!(string)(f); }

string toElem(string s) { return s; }

string toElem(uint x) { return uintToString(x); }

void \_fromElem(xml.Element e, ref string x) { x = e.text(); }

void \_fromElem(xml.Element e, ref bool x) { x = e.text() == "1"; }

void \_fromElem(xml.Element e, ref ubyte x) { x = toUbyte(e.text()); }

void \_fromElem(xml.Element e, ref uint x) { x = toUbyte(e.text()); }

void \_fromElem(xml.Element e, ref float x) { x = toFloat(e.text()); }

void fromElem(T)(xml.Element e, string s, ref T x)

{

if(xml.Element el = xml.getElement(e, s))

\_fromElem(el, x);

}

enum Compiler

{

DMD,

GDC,

LDC

}

enum OutputType

{

Executable,

StaticLib,

DLL

};

enum Subsystem

{

NotSet,

Console,

Windows,

Native,

Posix

};

enum CRuntime

{

None,

StaticRelease,

StaticDebug,

DynamicRelease,

DynamicDebug,

}

class ProjectOptions

{

bool obj;                // write object file

bool link;                // perform link

ubyte lib;                // write library file instead of object file(s) (1: static, 2:dynamic)

ubyte subsystem;

bool multiobj;                // break one object file into multiple ones

bool oneobj;                // write one object file instead of multiple ones

bool mscoff;                // use mscoff object files for Win32

bool trace;                // insert profiling hooks

bool quiet;                // suppress non-error messages

bool verbose;                // verbose compile

bool vtls;                // identify thread local variables

bool vgc;                // List all gc allocations including hidden ones (DMD 2.066+)

ubyte symdebug;                // insert debug symbolic information (0: none, 1: mago, 2: VS, 3: as debugging)

bool optimize;                // run optimizer

ubyte cpu;                // target CPU

bool isX86\_64;                // generate X86\_64 bit code

bool isLinux;                // generate code for linux

bool isOSX;                // generate code for Mac OSX

bool isWindows;                // generate code for Windows

bool isFreeBSD;                // generate code for FreeBSD

bool isSolaris;                // generate code for Solaris

bool scheduler;                // which scheduler to use

bool useDeprecated;        // allow use of deprecated features

bool errDeprecated;        // error when using deprecated features (2.061+)

bool useAssert;                // generate runtime code for assert()'s

bool useInvariants;        // generate class invariant checks

bool useIn;                // generate precondition checks

bool useOut;                // generate postcondition checks

ubyte useArrayBounds;        // 0: no array bounds checks

// 1: array bounds checks for safe functions only

// 2: array bounds checks for all functions

bool noboundscheck;        // no array bounds checking at all

bool useSwitchError;        // check for switches without a default

bool useUnitTests;        // generate unittest code

bool useInline;                // inline expand functions

ubyte release;                // build release version (0: -debug, 1: -release, 2: default)

bool preservePaths;        // !=0 means don't strip path from source file

bool warnings;                // enable warnings

bool infowarnings;        // enable informational warnings

bool checkProperty;        // enforce property syntax

bool genStackFrame;        // always generate stack frame

bool pic;                // generate position-independent-code for shared libs

bool cov;                // generate code coverage data

bool nofloat;                // code should not pull in floating point support

bool ignoreUnsupportedPragmas;        // rather than error on them

bool allinst;                // generate code for all template instantiations

bool stackStomp;        // add stack stomp code

float Dversion;                // D version number

ubyte compiler;                // 0: DMD, 1: GDC, 2:LDC

bool otherDMD;                // use explicit program path

bool ccTransOpt;        // translate D options to C where applicable

string cccmd;                // C/C++ compiler command prefix

string program;                // program name

string imppath;                // array of char\*'s of where to look for import modules

string fileImppath;        // array of char\*'s of where to look for file import modules

string outdir;                // target output directory

string objdir;                // .obj/.lib file output directory

string objname;                // .obj file output name

string libname;                // .lib file output name

bool doDocComments;        // process embedded documentation comments

string docdir;                // write documentation file to docdir directory

string docname;                // write documentation file to docname

string ddocfiles;        // macro include files for Ddoc

string modules\_ddoc; // generate modules.ddoc for candydoc

bool doHdrGeneration;        // process embedded documentation comments

string hdrdir;                // write 'header' file to docdir directory

string hdrname;                // write 'header' file to docname

bool doXGeneration;        // write JSON file

string xfilename;        // write JSON file to xfilename

uint debuglevel;        // debug level

string debugids;        // debug identifiers

uint versionlevel;        // version level

string versionids;        // version identifiers

bool dump\_source;

uint mapverbosity;

bool createImplib;

string defaultlibname;        // default library for non-debug builds

string debuglibname;        // default library for debug builds

string moduleDepsFile;        // filename for deps output

bool run;                // run resulting executable

string runargs;                // arguments for executable

bool runCv2pdb;                // run cv2pdb on executable

bool cv2pdbPre2043;                // pass version before 2.043 for older aa implementation

bool cv2pdbNoDemangle;        // do not demangle symbols

bool cv2pdbEnumType;        // use enumerator type

string pathCv2pdb;        // exe path for cv2pdb

string cv2pdbOptions;        // more options for cv2pdb

enum

{

kCombinedCompileAndLink,

kSingleFileCompilation,

kSeparateCompileAndLink,

kSeparateCompileOnly,

}

uint compilationModel = kCombinedCompileAndLink;

// Linker stuff

string objfiles;

string linkswitches;

string libfiles;

string libpaths;

string deffile;

string resfile;

string exefile;

bool useStdLibPath;

uint cRuntime;

bool privatePhobos;

string additionalOptions;

string preBuildCommand;

string postBuildCommand;

// debug options

string debugtarget;

string debugarguments;

string debugworkingdir;

bool debugattach;

string debugremote;

ubyte debugEngine; // 0: mixed, 1: mago, 2: native

bool debugStdOutToOutputWindow;

bool pauseAfterRunning;

string filesToClean;

this(bool dbg, bool x64)

{

Dversion = 2.043;

exefile = "$(OutDir)[\\$(ProjectName).exe](file:///\\$(ProjectName).exe)";

outdir = "$(ConfigurationName)";

objdir = "$(OutDir)";

debugtarget = "$(TARGETPATH)";

pathCv2pdb = "$(VisualDInstallDir)cv2pdb\\cv2pdb.exe";

program = "$(DMDInstallDir)windows\\bin\\dmd.exe";

xfilename = "$(IntDir)[\\$(TargetName).json](file:///\\$(TargetName).json)";

cccmd = "$(CC) -c";

ccTransOpt = true;

doXGeneration = true;

useStdLibPath = true;

cRuntime = CRuntime.StaticRelease;

debugEngine = 1;

filesToClean = "\*.obj;\*.cmd;\*.build;\*.json;\*.dep";

setDebug(dbg);

setX64(x64);

}

void setDebug(bool dbg)

{

runCv2pdb = dbg;

symdebug = dbg ? 3 : 0;

release = dbg ? 0 : 1;

optimize = release == 1;

useInline = release == 1;

}

void setX64(bool x64)

{

isX86\_64 = x64;

if(release != 1 && cRuntime == CRuntime.StaticRelease)

cRuntime = CRuntime.StaticDebug;

}

string objectFileExtension() { return compiler != Compiler.GDC ? "obj" : "o"; }

string otherCompilerPath() { return otherDMD ? program : null; }

bool useMSVCRT()

{

return (compiler == Compiler.DMD && (isX86\_64 || mscoff)) ||

(compiler == Compiler.LDC);

}

@property ref CompilerDirectories compilerDirectories()

{

switch(compiler)

{

default:

case Compiler.DMD: return Package.GetGlobalOptions().DMD;

case Compiler.GDC: return Package.GetGlobalOptions().GDC;

case Compiler.LDC: return Package.GetGlobalOptions().LDC;

}

}

bool isLDCforMinGW()

{

if (compiler != Compiler.LDC)

return false;

string installdir = Package.GetGlobalOptions().LDC.InstallDir;

if (installdir.empty)

return false;

return std.file.exists(normalizeDir(installdir) ~ "lib/libphobos2-ldc.a");

}

// common options with building phobos.lib

string dmdCommonCompileOptions()

{

string cmd;

if(isX86\_64)

cmd ~= " -m64";

else if(mscoff)

cmd ~= " -m32mscoff";

if(verbose)

cmd ~= " -v";

if(Dversion >= 2 && vtls)

cmd ~= " -vtls";

if(Dversion >= 2 && vgc)

cmd ~= " -vgc";

int symdbg = symdebug;

if(symdebug == 3)

symdbg = debugEngine == 1 ? 1 : 2;

if(symdbg == 1)

cmd ~= " -g";

if(symdbg == 2)

cmd ~= " -gc";

if(optimize)

cmd ~= " -O";

if(useDeprecated)

cmd ~= " -d";

if(useInline)

cmd ~= " -inline";

if(release == 1)

cmd ~= " -release";

else if(release == 0)

cmd ~= " -debug";

if(warnings)

cmd ~= " -w";

if(infowarnings)

cmd ~= " -wi";

if(checkProperty)

cmd ~= " -property";

if(genStackFrame)

cmd ~= " -gs";

if(stackStomp)

cmd ~= " -gx";

return cmd;

}

string buildDMDCommandLine(bool compile = true, bool performLink = true, bool deps = true, bool syntaxOnly = false)

{

string cmd;

if(otherDMD && program.length)

cmd = quoteNormalizeFilename(program);

else

cmd = "dmd";

if(performLink && Package.GetGlobalOptions().demangleError)

cmd = "\"$(VisualDInstallDir)pipedmd.exe\" " ~ cmd;

cmd ~= dmdCommonCompileOptions();

if(lib == OutputType.StaticLib && performLink)

cmd ~= " -lib";

if(multiobj)

cmd ~= " -multiobj";

if(trace)

cmd ~= " -profile";

if(quiet)

cmd ~= " -quiet";

else if(errDeprecated)

cmd ~= " -de";

if(Dversion >= 2 && noboundscheck)

cmd ~= " -noboundscheck";

if(useUnitTests)

cmd ~= " -unittest";

if(preservePaths)

cmd ~= " -op";

if(cov)

cmd ~= " -cov";

if(nofloat)

cmd ~= " -nofloat";

if(ignoreUnsupportedPragmas)

cmd ~= " -ignore";

if(allinst)

cmd ~= " -allinst";

if(privatePhobos)

cmd ~= " -defaultlib=" ~ quoteFilename(normalizeDir(outdir) ~ "privatephobos.lib");

if(doDocComments && compile && !syntaxOnly)

{

cmd ~= " -D";

if(docdir.length)

cmd ~= " -Dd" ~ quoteNormalizeFilename(docdir);

if(docname.length)

cmd ~= " -Df" ~ quoteNormalizeFilename(docname);

}

if(doHdrGeneration && compile && !syntaxOnly)

{

cmd ~= " -H";

if(hdrdir.length)

cmd ~= " -Hd" ~ quoteNormalizeFilename(hdrdir);

if(hdrname.length)

cmd ~= " -Hf" ~ quoteNormalizeFilename(hdrname);

}

if(doXGeneration && compile && !syntaxOnly)

{

cmd ~= " -X";

if(xfilename.length)

cmd ~= " -Xf" ~ quoteNormalizeFilename(xfilename);

}

string[] imports = tokenizeArgs(imppath);

foreach(imp; imports)

if(strip(imp).length)

cmd ~= " -I" ~ quoteNormalizeFilename(strip(imp));

string[] globalimports = tokenizeArgs(compilerDirectories.ImpSearchPath);

foreach(gimp; globalimports)

if(strip(gimp).length)

cmd ~= " -I" ~ quoteNormalizeFilename(strip(gimp));

string[] fileImports = tokenizeArgs(fileImppath);

foreach(imp; fileImports)

if(strip(imp).length)

cmd ~= " -J" ~ quoteNormalizeFilename(strip(imp));

string[] versions = tokenizeArgs(versionids);

foreach(ver; versions)

if(strip(ver).length)

cmd ~= " -version=" ~ strip(ver);

string[] ids = tokenizeArgs(debugids);

foreach(id; ids)

if(strip(id).length)

cmd ~= " -debug=" ~ strip(id);

if(deps && !syntaxOnly)

cmd ~= " -deps=" ~ quoteNormalizeFilename(getDependenciesPath());

if(performLink)

cmd ~= linkCommandLine();

return cmd;

}

string buildGDCCommandLine(bool compile = true, bool performLink = true, bool deps = true, bool syntaxOnly = false)

{

string cmd;

if(otherDMD && program.length)

cmd = quoteNormalizeFilename(program);

else

cmd = "gdc";

if(performLink && Package.GetGlobalOptions().demangleError)

cmd = "\"$(VisualDInstallDir)pipedmd.exe\" -gdcmode " ~ cmd;

//                if(lib && performLink)

//                        cmd ~= " -lib";

//                if(multiobj)

//                        cmd ~= " -multiobj";

if(lib == OutputType.DLL)

cmd ~= " -mdll";

if(subsystem == Subsystem.Windows)

cmd ~= " -mwindows";

else if(subsystem == Subsystem.Console)

cmd ~= " -mconsole";

if(isX86\_64)

cmd ~= " -m64";

else

cmd ~= " -m32";

if(trace)

cmd ~= " -pg";

//                if(quiet)

//                        cmd ~= " -quiet";

if(verbose)

cmd ~= " -fd-verbose";

if(Dversion < 2)

cmd ~= " -fd-version=1";

if(Dversion >= 2 && vtls)

cmd ~= " -fd-vtls";

if(Dversion >= 2 && vgc)

cmd ~= " -fd-vgc";

if(symdebug > 0)

cmd ~= " -g";

//if(symdebug == 2)

// cmd ~= " -fdebug-c";

if(optimize)

cmd ~= " -O3";

if(useDeprecated)

cmd ~= " -fdeprecated";

if(Dversion >= 2 && noboundscheck)

cmd ~= " -fno-bounds-check";

if(useUnitTests)

cmd ~= " -funittest";

if(!useInline)

cmd ~= " -fno-inline-functions";

if(release == 1)

cmd ~= " -frelease";

else if (release == 0)

cmd ~= " -fdebug";

//                if(preservePaths)

//                        cmd ~= " -op";

if(warnings)

cmd ~= " -Werror";

if(infowarnings)

cmd ~= " -Wall";

if(checkProperty)

cmd ~= " -fproperty";

if(genStackFrame)

cmd ~= " -fno-omit-frame-pointer";

if(cov)

cmd ~= " -fprofile-arcs -ftest-coverage";

//                if(nofloat)

//                        cmd ~= " -nofloat";

if(ignoreUnsupportedPragmas)

cmd ~= " -fignore-unknown-pragmas";

if(doDocComments && compile && !syntaxOnly)

{

cmd ~= " -fdoc";

if(docdir.length)

cmd ~= " -fdoc-dir=" ~ quoteNormalizeFilename(docdir);

if(docname.length)

cmd ~= " -fdoc-file=" ~ quoteNormalizeFilename(docname);

}

if(doHdrGeneration && compile && !syntaxOnly)

{

cmd ~= " -fintfc";

if(hdrdir.length)

cmd ~= " -fintfc-dir=" ~ quoteNormalizeFilename(hdrdir);

if(hdrname.length)

cmd ~= " -fintfc-file=" ~ quoteNormalizeFilename(hdrname);

}

if(doXGeneration && compile && !syntaxOnly)

{

string xfile = xfilename.length ? xfilename : "$(OUTDIR)[\\$(SAFEPROJECTNAME).json](file:///\\$(SAFEPROJECTNAME).json)";

cmd ~= " -fXf=" ~ quoteNormalizeFilename(xfile);

}

string[] imports = tokenizeArgs(imppath);

foreach(imp; imports)

if(strip(imp).length)

cmd ~= " -I" ~ quoteNormalizeFilename(strip(imp));

string[] globalimports = tokenizeArgs(compilerDirectories.ImpSearchPath);

foreach(gimp; globalimports)

if(strip(gimp).length)

cmd ~= " -I" ~ quoteNormalizeFilename(strip(gimp));

string[] fileImports = tokenizeArgs(fileImppath);

foreach(imp; fileImports)

if(strip(imp).length)

cmd ~= " -J" ~ quoteNormalizeFilename(strip(imp));

string[] versions = tokenizeArgs(versionids);

foreach(ver; versions)

if(strip(ver).length)

cmd ~= " -fversion=" ~ strip(ver);

string[] ids = tokenizeArgs(debugids);

foreach(id; ids)

if(strip(id).length)

cmd ~= " -fdebug=" ~ strip(id);

if(deps && !syntaxOnly)

cmd ~= " -fdeps=" ~ quoteNormalizeFilename(getDependenciesPath());

if(performLink)

cmd ~= linkCommandLine();

return cmd;

}

string buildLDCCommandLine(bool compile = true, bool performLink = true, bool deps = true, bool syntaxOnly = false)

{

string cmd;

if(otherDMD && program.length)

cmd = quoteNormalizeFilename(program);

else

cmd = "ldc2";

if(performLink && Package.GetGlobalOptions().demangleError)

cmd = "\"$(VisualDInstallDir)pipedmd.exe\" " ~ cmd;

if(lib == OutputType.StaticLib && performLink)

cmd ~= " -lib";

if(isX86\_64)

cmd ~= " -m64";

else

cmd ~= " -m32";

if(verbose)

cmd ~= " -v";

int symdbg = symdebug;

if(symdebug == 3)

symdbg = debugEngine == 1 ? 1 : 2;

if(symdbg == 1)

cmd ~= " -g";

if(symdbg == 2)

cmd ~= " -gc";

if(optimize)

cmd ~= " -O";

if(useDeprecated)

cmd ~= " -d";

else if(errDeprecated)

cmd ~= " -de";

if(useUnitTests)

cmd ~= " -unittest";

if(release == 1)

cmd ~= " -release";

else if (release == 0)

cmd ~= " -d-debug";

if(preservePaths)

cmd ~= " -op";

if(warnings)

cmd ~= " -w";

if(infowarnings)

cmd ~= " -wi";

if(checkProperty)

cmd ~= " -property";

if(ignoreUnsupportedPragmas)

cmd ~= " -ignore";

if(doDocComments && compile && !syntaxOnly)

{

cmd ~= " -D";

if(docdir.length)

cmd ~= " -Dd=" ~ quoteNormalizeFilename(docdir);

if(docname.length)

cmd ~= " -Df=" ~ quoteNormalizeFilename(docname);

}

if(doHdrGeneration && compile && !syntaxOnly)

{

cmd ~= " -H";

if(hdrdir.length)

cmd ~= " -Hd=" ~ quoteNormalizeFilename(hdrdir);

if(hdrname.length)

cmd ~= " -Hf=" ~ quoteNormalizeFilename(hdrname);

}

if(doXGeneration && compile && !syntaxOnly)

{

cmd ~= " -X";

if(xfilename.length)

cmd ~= " -Xf=" ~ quoteNormalizeFilename(xfilename);

}

string[] imports = tokenizeArgs(imppath);

foreach(imp; imports)

if(strip(imp).length)

cmd ~= " -I=" ~ quoteNormalizeFilename(strip(imp));

string[] globalimports = tokenizeArgs(compilerDirectories.ImpSearchPath);

foreach(gimp; globalimports)

if(strip(gimp).length)

cmd ~= " -I=" ~ quoteNormalizeFilename(strip(gimp));

string[] fileImports = tokenizeArgs(fileImppath);

foreach(imp; fileImports)

if(strip(imp).length)

cmd ~= " -J=" ~ quoteNormalizeFilename(strip(imp));

string[] versions = tokenizeArgs(versionids);

foreach(ver; versions)

if(strip(ver).length)

cmd ~= " -d-version=" ~ strip(ver);

string[] ids = tokenizeArgs(debugids);

foreach(id; ids)

if(strip(id).length)

cmd ~= " -d-debug=" ~ strip(id);

if(deps && !syntaxOnly)

cmd ~= " -deps=" ~ quoteNormalizeFilename(getDependenciesPath());

if(performLink)

cmd ~= linkCommandLine();

return cmd;

}

string buildCommandLine(bool compile = true, bool performLink = true, bool deps = true, bool syntaxOnly = false)

{

if(compiler == Compiler.DMD)

return buildDMDCommandLine(compile, performLink, deps, syntaxOnly);

if(compiler == Compiler.LDC)

return buildLDCCommandLine(compile, performLink, deps, syntaxOnly);

if(!compile && performLink && lib == OutputType.StaticLib)

return buildARCommandLine();

return buildGDCCommandLine(compile, performLink, deps, syntaxOnly);

}

string buildARCommandLine()

{

string cmd = "ar cru " ~ quoteNormalizeFilename(getTargetPath());

return cmd;

}

string linkDMDCommandLine(bool mslink)

{

string cmd;

string dmdoutfile = getTargetPath();

if(usesCv2pdb())

dmdoutfile ~= "\_cv";

cmd ~= getOutputFileOption(dmdoutfile);

if(mslink && compiler != Compiler.DMD)

cmd ~= " -L/MAP:\"$(INTDIR)\\$(SAFEPROJECTNAME).map\"";

else

cmd ~= " -map \"$(INTDIR)[\\$(SAFEPROJECTNAME).map\](file:///\\$(SAFEPROJECTNAME).map\)"";

switch(mapverbosity)

{

case 0: cmd ~= mslink ? "" : " -L/NOMAP"; break; // actually still creates map file

case 1: cmd ~= mslink ? "-L/MAPINFO:EXPORTS" : " -L/MAP:ADDRESS"; break;

case 2: break;

case 3: cmd ~= mslink ? "-L/MAPINFO:EXPORTS,LINES" : " -L/MAP:FULL"; break;

case 4: cmd ~= mslink ? "-L/MAPINFO:EXPORTS,LINES,FIXUPS" : " -L/MAP:FULL -L/XREF"; break;

default: break;

}

if(lib != OutputType.StaticLib)

{

if(createImplib)

cmd ~= " -L/IMPLIB:$(OUTDIR)\\$(PROJECTNAME).lib";

if(objfiles.length)

cmd ~= " " ~ objfiles;

if(deffile.length)

cmd ~= " " ~ deffile;

if(libfiles.length)

cmd ~= " " ~ libfiles;

if(resfile.length)

cmd ~= " " ~ resfile;

switch(subsystem)

{

default:

case Subsystem.NotSet: break;

case Subsystem.Console: cmd ~= " -L/SUBSYSTEM:CONSOLE"; break;

case Subsystem.Windows: cmd ~= " -L/SUBSYSTEM:WINDOWS"; break;

case Subsystem.Native: cmd ~= " -L/SUBSYSTEM:NATIVE"; break;

case Subsystem.Posix: cmd ~= " -L/SUBSYSTEM:POSIX"; break;

}

}

return cmd;

}

string linkGDCCommandLine()

{

string cmd;

string linkeropt = " -Wl,";

string dmdoutfile = getTargetPath();

if(usesCv2pdb())

dmdoutfile ~= "\_cv";

cmd ~= " -o " ~ quoteNormalizeFilename(dmdoutfile);

switch(mapverbosity)

{

case 0: // no map

break;

default:

cmd ~= linkeropt ~ "-Map=\"$(INTDIR)[\\$(SAFEPROJECTNAME).map\](file:///\\$(SAFEPROJECTNAME).map\)"";

break;

}

string[] lpaths = tokenizeArgs(libpaths);

if(useStdLibPath)

lpaths ~= tokenizeArgs(isX86\_64 ? compilerDirectories.LibSearchPath64 : compilerDirectories.LibSearchPath);

else

cmd ~= linkeropt ~ "-nostdlib";

foreach(lp; lpaths)

cmd ~= linkeropt ~ "-L," ~ quoteFilename(lp);

if(lib != OutputType.StaticLib)

{

//                        if(createImplib)

//                                cmd ~= " -L/IMPLIB:$(OUTDIR)\\$(PROJECTNAME).lib";

if(objfiles.length)

cmd ~= " " ~ objfiles;

if(deffile.length)

cmd ~= " " ~ deffile;

// added later in getCommandFileList

//                        if(libfiles.length)

//                                cmd ~= " " ~ libfiles;

if(resfile.length)

cmd ~= " " ~ resfile;

}

return cmd;

}

string linkLDCCommandLine()

{

string cmd;

string linkeropt = " -L=";

string dmdoutfile = getTargetPath();

if(usesCv2pdb())

dmdoutfile ~= "\_cv";

cmd ~= " -of=" ~ quoteNormalizeFilename(dmdoutfile);

switch(mapverbosity)

{

case 0: // no map

break;

default:

cmd ~= linkeropt ~ "-Map=\"$(INTDIR)[\\$(SAFEPROJECTNAME).map\](file:///\\$(SAFEPROJECTNAME).map\)"";

break;

}

string[] lpaths = tokenizeArgs(libpaths);

if(useStdLibPath)

lpaths ~= tokenizeArgs(isX86\_64 ? compilerDirectories.LibSearchPath64 : compilerDirectories.LibSearchPath);

else

cmd ~= linkeropt ~ "-nostdlib";

foreach(lp; lpaths)

cmd ~= linkeropt ~ "-L," ~ quoteFilename(lp);

if(lib != OutputType.StaticLib)

{

//                        if(createImplib)

//                                cmd ~= " -L/IMPLIB:$(OUTDIR)\\$(PROJECTNAME).lib";

if(objfiles.length)

cmd ~= " " ~ objfiles;

if(deffile.length)

cmd ~= " " ~ deffile;

// added later in getCommandFileList

//                        if(libfiles.length)

//                                cmd ~= " " ~ libfiles;

if(resfile.length)

cmd ~= " " ~ resfile;

}

return cmd;

}

string optlinkCommandLine(string[] lnkfiles, string inioptions, string workdir, bool mslink)

{

string cmd;

string dmdoutfile = getTargetPath();

if(usesCv2pdb())

dmdoutfile ~= "\_cv";

string mapfile = "\"$(INTDIR)[\\$(SAFEPROJECTNAME).map\](file:///\\$(SAFEPROJECTNAME).map\)"";

string plus = mslink ? " " : "+";

static string plusList(string[] lnkfiles, string ext, string sep)

{

if(ext.length == 0 || ext[0] != '.')

ext = "." ~ ext;

string s;

foreach(i, file; lnkfiles)

{

if(toLower(extension(file)) != ext)

continue;

if(s.length > 0)

s ~= sep;

s ~= quoteNormalizeFilename(file);

}

return s;

}

inioptions ~= " " ~ additionalOptions.replace("\n", " ");

string[] opts = tokenizeArgs(inioptions, false);

opts = expandResponseFiles(opts, workdir);

string addopts;

foreach(ref opt; opts)

{

opt = unquoteArgument(opt);

if(opt.startsWith("-L"))

addopts ~= " " ~ quoteFilename(opt[2..$]);

if(opt[0] != '-')

lnkfiles ~= opt;

}

cmd ~= plusList(lnkfiles, objectFileExtension(), plus);

cmd ~= mslink ? " /OUT:" : ",";

cmd ~= quoteNormalizeFilename(dmdoutfile);

cmd ~= mslink ? " /MAP:" : ",";

cmd ~= mapfile;

cmd ~= mslink ? " " : ",";

string[] libs = tokenizeArgs(libfiles);

libs ~= "user32.lib";

libs ~= "kernel32.lib";

if(useMSVCRT())

if(std.file.exists(Package.GetGlobalOptions().VCInstallDir ~ "lib\\legacy\_stdio\_definitions.lib"))

libs ~= "legacy\_stdio\_definitions.lib";

cmd ~= plusList(lnkfiles ~ libs, ".lib", plus);

string[] lpaths = tokenizeArgs(libpaths);

if(useStdLibPath)

lpaths ~= tokenizeArgs(isX86\_64 ? compilerDirectories.LibSearchPath64 :

mscoff ? compilerDirectories.LibSearchPath32coff : compilerDirectories.LibSearchPath);

foreach(lp; lpaths)

if(mslink)

cmd ~= " /LIBPATH:" ~ quoteFilename(normalizeDir(unquoteArgument(lp))[0..$-1]); // avoid trailing \ for quoted files

else

cmd ~= "+" ~ quoteFilename(normalizeDir(unquoteArgument(lp))); // optlink needs trailing \

string def = deffile.length ? quoteNormalizeFilename(deffile) : plusList(lnkfiles, ".def", mslink ? " /DEF:" : plus);

string res = resfile.length ? quoteNormalizeFilename(resfile) : plusList(lnkfiles, ".res", plus);

if(mslink)

{

if(def.length)

cmd ~= " /DEF:" ~ def;

if(res.length)

cmd ~= " " ~ res;

}

else

{

if(def.length || res.length)

cmd ~= "," ~ def;

if(res.length)

cmd ~= "," ~ res;

}

// options

// "/m" to geneate map?

if(!mslink)

switch(mapverbosity)

{

case 0: cmd ~= "/NOMAP"; break; // actually still creates map file

case 1: cmd ~= "/MAP:ADDRESS"; break;

case 2: break;

case 3: cmd ~= "/MAP:FULL"; break;

case 4: cmd ~= "/MAP:FULL/XREF"; break;

default: break;

}

if(symdebug)

cmd ~= mslink ? " /DEBUG" : "/CO";

cmd ~= mslink ? " /INCREMENTAL:NO /NOLOGO" : "/NOI/DELEXE";

if(mslink)

{

switch(cRuntime)

{

case CRuntime.None: cmd ~= " /NODEFAULTLIB:libcmt"; break;

case CRuntime.StaticRelease: break;

case CRuntime.StaticDebug: cmd ~= " /NODEFAULTLIB:libcmt libcmtd.lib"; break;

case CRuntime.DynamicRelease: cmd ~= " /NODEFAULTLIB:libcmt msvcrt.lib"; break;

case CRuntime.DynamicDebug: cmd ~= " /NODEFAULTLIB:libcmt msvcrtd.lib"; break;

default: break;

}

}

if(lib != OutputType.StaticLib)

{

if(createImplib)

cmd ~= " /IMPLIB:$(OUTDIR)\\$(PROJECTNAME).lib";

switch(subsystem)

{

default:

case Subsystem.NotSet: break;

case Subsystem.Console: cmd ~= " /SUBSYSTEM:CONSOLE"; break;

case Subsystem.Windows: cmd ~= " /SUBSYSTEM:WINDOWS"; break;

case Subsystem.Native: cmd ~= " /SUBSYSTEM:NATIVE"; break;

case Subsystem.Posix: cmd ~= " /SUBSYSTEM:POSIX"; break;

}

}

cmd ~= addopts;

return cmd;

}

string linkCommandLine()

{

if(compiler == Compiler.GDC)

return linkGDCCommandLine();

else if(isLDCforMinGW())

return linkLDCCommandLine();

else if(compiler == Compiler.LDC)

return linkDMDCommandLine(true); // MS link

else

return linkDMDCommandLine(isX86\_64);

}

string getOutputDirOption()

{

switch(compiler)

{

default:

case Compiler.DMD: return " -od" ~ quoteFilename(objdir);

case Compiler.LDC: return " -od=" ~ quoteFilename(objdir);

case Compiler.GDC: return ""; // does not work with GDC

}

}

string getOutputFileOption(string file)

{

switch(compiler)

{

default:

case Compiler.DMD: return " -of" ~ quoteFilename(file);

case Compiler.LDC: return " -of=" ~ quoteFilename(file);

case Compiler.GDC: return " -o " ~ quoteFilename(file);

}

}

string getCppCommandLine(string file, bool setenv)

{

int cc; // 0-3 for dmc,cl,clang,gdc

switch(compiler)

{

default:

case Compiler.DMD: cc = (isX86\_64 || mscoff ? 1 : 0); break;

case Compiler.LDC: cc = (isLDCforMinGW() ? 2 : 1); break;

case Compiler.GDC: cc = 3; break;

}

string cmd = cccmd;

if(cc == 1 && setenv)

cmd = `call "%VCINSTALLDIR%\vcvarsall.bat" ` ~ (isX86\_64 ? "x86\_amd64" : "x86") ~ "\n" ~ cmd;

static string[4] outObj = [ " -o", " -Fo", " -o", " -o " ];

if (file.length)

cmd ~= outObj[cc] ~ quoteFilename(file);

if (!ccTransOpt)

return cmd;

static string[4] dbg = [ " -g", " -Z7", " -g", " -g" ];

if(symdebug)

cmd ~= dbg[cc];

if (release == 1)

cmd ~= " -DNDEBUG";

static string[4] opt = [ " -O", " -Ox", " -O3", " -O3" ];

if(optimize)

cmd ~= opt[cc];

if (quiet && cc == 1)

cmd ~= " /NOLOGO";

return cmd;

}

string getDependenciesFileOption(string file)

{

if(compiler == Compiler.GDC)

return " -fdeps=" ~ quoteFilename(file);

else

return " -deps=" ~ quoteFilename(file);

}

string getAdditionalLinkOptions()

{

if(compiler != Compiler.DMD && lib == OutputType.StaticLib)

return ""; // no options to ar

return additionalOptions.replace("\n", " "); // always filtered through compiler

}

string getTargetPath()

{

if(exefile.length)

return normalizePath(exefile);

if(lib == OutputType.StaticLib)

return "$(OutDir)[\\$(ProjectName).lib](file:///\\$(ProjectName).lib)";

return "$(OutDir)[\\$(ProjectName).exe](file:///\\$(ProjectName).exe)";

}

string getDependenciesPath()

{

return normalizeDir(objdir) ~ "$(ProjectName).dep";

}

string getCommandLinePath()

{

return normalizeDir(objdir) ~ "$(ProjectName)." ~ kCmdLogFileExtension;

}

// "linking" includes building library (through ar with GDC, internal with DMD)

bool doSeparateLink()

{

if(compilationModel == ProjectOptions.kSeparateCompileOnly)

return false;

bool separateLink = compilationModel == ProjectOptions.kSeparateCompileAndLink;

if (compiler == Compiler.GDC && lib == OutputType.StaticLib)

separateLink = true;

if (compiler == Compiler.DMD && lib != OutputType.StaticLib)

{

if(Package.GetGlobalOptions().optlinkDeps)

separateLink = true;

else if(isX86\_64 && Package.GetGlobalOptions().DMD.overrideIni64)

separateLink = true;

else if(!isX86\_64 && mscoff && Package.GetGlobalOptions().DMD.overrideIni32coff)

separateLink = true;

}

return separateLink;

}

bool callLinkerDirectly()

{

bool dmdlink = compiler == Compiler.DMD && doSeparateLink() && lib != OutputType.StaticLib;

return dmdlink; // && !isX86\_64;

}

bool usesCv2pdb()

{

if(compiler == Compiler.DMD && (isX86\_64 || mscoff))

return false; // should generate correct debug info directly

return (/\*compiler == Compiler.DMD && \*/symdebug && runCv2pdb && lib != OutputType.StaticLib && debugEngine != 1);

}

string appendCv2pdb()

{

if(usesCv2pdb())

{

string target = getTargetPath();

string cmd = quoteFilename(pathCv2pdb);

if(Dversion < 2)

cmd ~= " -D" ~ to!(string)(Dversion) ~ " ";

else if(cv2pdbPre2043)

cmd ~= " -D2.001";

if(cv2pdbEnumType)

cmd ~= " -e";

if(cv2pdbNoDemangle)

cmd ~= " -n";

if(cv2pdbOptions.length)

cmd ~= " " ~ cv2pdbOptions;

cmd ~= " " ~ quoteFilename(target ~ "\_cv") ~ " " ~ quoteFilename(target);

return cmd;

}

return "";

}

string replaceEnvironment(string cmd, Config config, string inputfile = "", string outputfile = "")

{

if(indexOf(cmd, '$') < 0)

return cmd;

string configname = config.mName;

string projectpath = config.GetProjectPath();

string safeprojectpath = projectpath.replace(" ", "\_");

string[string] replacements;

string solutionpath = GetSolutionFilename();

if(solutionpath.length)

addFileMacros(solutionpath, "SOLUTION", replacements);

replacements["PLATFORMNAME"] = config.mPlatform;

addFileMacros(projectpath, "PROJECT", replacements);

replacements["PROJECTNAME"] = config.GetProjectName();

addFileMacros(safeprojectpath, "SAFEPROJECT", replacements);

replacements["SAFEPROJECTNAME"] = config.GetProjectName().replace(" ", "\_");

addFileMacros(inputfile.length ? inputfile : projectpath, "INPUT", replacements);

replacements["CONFIGURATIONNAME"] = configname;

replacements["CONFIGURATION"] = configname;

replacements["OUTDIR"] = normalizePath(outdir);

replacements["INTDIR"] = normalizePath(objdir);

Package.GetGlobalOptions().addReplacements(replacements);

replacements["CC"] = config.GetCppCompiler();

string targetpath = outputfile.length ? outputfile : getTargetPath();

string target = replaceMacros(targetpath, replacements);

addFileMacros(target, "TARGET", replacements);

return replaceMacros(cmd, replacements);

}

void writeXML(xml.Element elem)

{

elem ~= new xml.Element("obj", toElem(obj));

elem ~= new xml.Element("link", toElem(link));

elem ~= new xml.Element("lib", toElem(lib));

elem ~= new xml.Element("subsystem", toElem(subsystem));

elem ~= new xml.Element("multiobj", toElem(multiobj));

elem ~= new xml.Element("singleFileCompilation", toElem(compilationModel));

elem ~= new xml.Element("oneobj", toElem(oneobj));

elem ~= new xml.Element("mscoff", toElem(mscoff));

elem ~= new xml.Element("trace", toElem(trace));

elem ~= new xml.Element("quiet", toElem(quiet));

elem ~= new xml.Element("verbose", toElem(verbose));

elem ~= new xml.Element("vtls", toElem(vtls));

elem ~= new xml.Element("vgc", toElem(vgc));

elem ~= new xml.Element("symdebug", toElem(symdebug));

elem ~= new xml.Element("optimize", toElem(optimize));

elem ~= new xml.Element("cpu", toElem(cpu));

elem ~= new xml.Element("isX86\_64", toElem(isX86\_64));

elem ~= new xml.Element("isLinux", toElem(isLinux));

elem ~= new xml.Element("isOSX", toElem(isOSX));

elem ~= new xml.Element("isWindows", toElem(isWindows));

elem ~= new xml.Element("isFreeBSD", toElem(isFreeBSD));

elem ~= new xml.Element("isSolaris", toElem(isSolaris));

elem ~= new xml.Element("scheduler", toElem(scheduler));

elem ~= new xml.Element("useDeprecated", toElem(useDeprecated));

elem ~= new xml.Element("errDeprecated", toElem(errDeprecated));

elem ~= new xml.Element("useAssert", toElem(useAssert));

elem ~= new xml.Element("useInvariants", toElem(useInvariants));

elem ~= new xml.Element("useIn", toElem(useIn));

elem ~= new xml.Element("useOut", toElem(useOut));

elem ~= new xml.Element("useArrayBounds", toElem(useArrayBounds));

elem ~= new xml.Element("noboundscheck", toElem(noboundscheck));

elem ~= new xml.Element("useSwitchError", toElem(useSwitchError));

elem ~= new xml.Element("useUnitTests", toElem(useUnitTests));

elem ~= new xml.Element("useInline", toElem(useInline));

elem ~= new xml.Element("release", toElem(release));

elem ~= new xml.Element("preservePaths", toElem(preservePaths));

elem ~= new xml.Element("warnings", toElem(warnings));

elem ~= new xml.Element("infowarnings", toElem(infowarnings));

elem ~= new xml.Element("checkProperty", toElem(checkProperty));

elem ~= new xml.Element("genStackFrame", toElem(genStackFrame));

elem ~= new xml.Element("pic", toElem(pic));

elem ~= new xml.Element("cov", toElem(cov));

elem ~= new xml.Element("nofloat", toElem(nofloat));

elem ~= new xml.Element("Dversion", toElem(Dversion));

elem ~= new xml.Element("ignoreUnsupportedPragmas", toElem(ignoreUnsupportedPragmas));

elem ~= new xml.Element("allinst", toElem(allinst));

elem ~= new xml.Element("stackStomp", toElem(stackStomp));

elem ~= new xml.Element("compiler", toElem(compiler));

elem ~= new xml.Element("otherDMD", toElem(otherDMD));

elem ~= new xml.Element("cccmd", toElem(cccmd));

elem ~= new xml.Element("ccTransOpt", toElem(ccTransOpt));

elem ~= new xml.Element("program", toElem(program));

elem ~= new xml.Element("imppath", toElem(imppath));

elem ~= new xml.Element("fileImppath", toElem(fileImppath));

elem ~= new xml.Element("outdir", toElem(outdir));

elem ~= new xml.Element("objdir", toElem(objdir));

elem ~= new xml.Element("objname", toElem(objname));

elem ~= new xml.Element("libname", toElem(libname));

elem ~= new xml.Element("doDocComments", toElem(doDocComments));

elem ~= new xml.Element("docdir", toElem(docdir));

elem ~= new xml.Element("docname", toElem(docname));

elem ~= new xml.Element("modules\_ddoc", toElem(modules\_ddoc));

elem ~= new xml.Element("ddocfiles", toElem(ddocfiles));

elem ~= new xml.Element("doHdrGeneration", toElem(doHdrGeneration));

elem ~= new xml.Element("hdrdir", toElem(hdrdir));

elem ~= new xml.Element("hdrname", toElem(hdrname));

elem ~= new xml.Element("doXGeneration", toElem(doXGeneration));

elem ~= new xml.Element("xfilename", toElem(xfilename));

elem ~= new xml.Element("debuglevel", toElem(debuglevel));

elem ~= new xml.Element("debugids", toElem(debugids));

elem ~= new xml.Element("versionlevel", toElem(versionlevel));

elem ~= new xml.Element("versionids", toElem(versionids));

elem ~= new xml.Element("dump\_source", toElem(dump\_source));

elem ~= new xml.Element("mapverbosity", toElem(mapverbosity));

elem ~= new xml.Element("createImplib", toElem(createImplib));

elem ~= new xml.Element("defaultlibname", toElem(defaultlibname));

elem ~= new xml.Element("debuglibname", toElem(debuglibname));

elem ~= new xml.Element("moduleDepsFile", toElem(moduleDepsFile));

elem ~= new xml.Element("run", toElem(run));

elem ~= new xml.Element("runargs", toElem(runargs));

elem ~= new xml.Element("runCv2pdb", toElem(runCv2pdb));

elem ~= new xml.Element("pathCv2pdb", toElem(pathCv2pdb));

elem ~= new xml.Element("cv2pdbPre2043", toElem(cv2pdbPre2043));

elem ~= new xml.Element("cv2pdbNoDemangle", toElem(cv2pdbNoDemangle));

elem ~= new xml.Element("cv2pdbEnumType", toElem(cv2pdbEnumType));

elem ~= new xml.Element("cv2pdbOptions", toElem(cv2pdbOptions));

// Linker stuff

elem ~= new xml.Element("objfiles", toElem(objfiles));

elem ~= new xml.Element("linkswitches", toElem(linkswitches));

elem ~= new xml.Element("libfiles", toElem(libfiles));

elem ~= new xml.Element("libpaths", toElem(libpaths));

elem ~= new xml.Element("deffile", toElem(deffile));

elem ~= new xml.Element("resfile", toElem(resfile));

elem ~= new xml.Element("exefile", toElem(exefile));

elem ~= new xml.Element("useStdLibPath", toElem(useStdLibPath));

elem ~= new xml.Element("cRuntime", toElem(cRuntime));

elem ~= new xml.Element("privatePhobos", toElem(privatePhobos));

elem ~= new xml.Element("additionalOptions", toElem(additionalOptions));

elem ~= new xml.Element("preBuildCommand", toElem(preBuildCommand));

elem ~= new xml.Element("postBuildCommand", toElem(postBuildCommand));

elem ~= new xml.Element("filesToClean", toElem(filesToClean));

}

void writeDebuggerXML(xml.Element elem)

{

elem ~= new xml.Element("debugtarget", toElem(debugtarget));

elem ~= new xml.Element("debugarguments", toElem(debugarguments));

elem ~= new xml.Element("debugworkingdir", toElem(debugworkingdir));

elem ~= new xml.Element("debugattach", toElem(debugattach));

elem ~= new xml.Element("debugremote", toElem(debugremote));

elem ~= new xml.Element("debugEngine", toElem(debugEngine));

elem ~= new xml.Element("debugStdOutToOutputWindow", toElem(debugStdOutToOutputWindow));

elem ~= new xml.Element("pauseAfterRunning", toElem(pauseAfterRunning));

}

void readXML(xml.Element elem)

{

fromElem(elem, "obj", obj);

fromElem(elem, "link", link);

fromElem(elem, "lib", lib);

fromElem(elem, "subsystem", subsystem);

fromElem(elem, "multiobj", multiobj);

fromElem(elem, "singleFileCompilation", compilationModel);

fromElem(elem, "oneobj", oneobj);

fromElem(elem, "mscoff", mscoff);

fromElem(elem, "trace", trace);

fromElem(elem, "quiet", quiet);

fromElem(elem, "verbose", verbose);

fromElem(elem, "vtls", vtls);

fromElem(elem, "vgc", vgc);

fromElem(elem, "symdebug", symdebug);

fromElem(elem, "optimize", optimize);

fromElem(elem, "cpu", cpu);

fromElem(elem, "isX86\_64", isX86\_64);

fromElem(elem, "isLinux", isLinux);

fromElem(elem, "isOSX", isOSX);

fromElem(elem, "isWindows", isWindows);

fromElem(elem, "isFreeBSD", isFreeBSD);

fromElem(elem, "isSolaris", isSolaris);

fromElem(elem, "scheduler", scheduler);

fromElem(elem, "useDeprecated", useDeprecated);

fromElem(elem, "errDeprecated", errDeprecated);

fromElem(elem, "useAssert", useAssert);

fromElem(elem, "useInvariants", useInvariants);

fromElem(elem, "useIn", useIn);

fromElem(elem, "useOut", useOut);

fromElem(elem, "useArrayBounds", useArrayBounds);

fromElem(elem, "noboundscheck", noboundscheck);

fromElem(elem, "useSwitchError", useSwitchError);

fromElem(elem, "useUnitTests", useUnitTests);

fromElem(elem, "useInline", useInline);

fromElem(elem, "release", release);

fromElem(elem, "preservePaths", preservePaths);

fromElem(elem, "warnings", warnings);

fromElem(elem, "infowarnings", infowarnings);

fromElem(elem, "checkProperty", checkProperty);

fromElem(elem, "genStackFrame", genStackFrame);

fromElem(elem, "pic", pic);

fromElem(elem, "cov", cov);

fromElem(elem, "nofloat", nofloat);

fromElem(elem, "Dversion", Dversion);

fromElem(elem, "ignoreUnsupportedPragmas", ignoreUnsupportedPragmas);

fromElem(elem, "allinst", allinst);

fromElem(elem, "stackStomp", stackStomp);

fromElem(elem, "compiler", compiler);

fromElem(elem, "otherDMD", otherDMD);

fromElem(elem, "cccmd", cccmd);

fromElem(elem, "ccTransOpt", ccTransOpt);

fromElem(elem, "program", program);

fromElem(elem, "imppath", imppath);

fromElem(elem, "fileImppath", fileImppath);

fromElem(elem, "outdir", outdir);

fromElem(elem, "objdir", objdir);

fromElem(elem, "objname", objname);

fromElem(elem, "libname", libname);

fromElem(elem, "doDocComments", doDocComments);

fromElem(elem, "docdir", docdir);

fromElem(elem, "docname", docname);

fromElem(elem, "modules\_ddoc", modules\_ddoc);

fromElem(elem, "ddocfiles", ddocfiles);

fromElem(elem, "doHdrGeneration", doHdrGeneration);

fromElem(elem, "hdrdir", hdrdir);

fromElem(elem, "hdrname", hdrname);

fromElem(elem, "doXGeneration", doXGeneration);

fromElem(elem, "xfilename", xfilename);

fromElem(elem, "debuglevel", debuglevel);

fromElem(elem, "debugids", debugids);

fromElem(elem, "versionlevel", versionlevel);

fromElem(elem, "versionids", versionids);

fromElem(elem, "dump\_source", dump\_source);

fromElem(elem, "mapverbosity", mapverbosity);

fromElem(elem, "createImplib", createImplib);

fromElem(elem, "defaultlibname", defaultlibname);

fromElem(elem, "debuglibname", debuglibname);

fromElem(elem, "moduleDepsFile", moduleDepsFile);

fromElem(elem, "run", run);

fromElem(elem, "runargs", runargs);

fromElem(elem, "runCv2pdb", runCv2pdb);

fromElem(elem, "pathCv2pdb", pathCv2pdb);

fromElem(elem, "cv2pdbPre2043", cv2pdbPre2043);

fromElem(elem, "cv2pdbNoDemangle", cv2pdbNoDemangle);

fromElem(elem, "cv2pdbEnumType", cv2pdbEnumType);

fromElem(elem, "cv2pdbOptions", cv2pdbOptions);

// Linker stuff

fromElem(elem, "objfiles", objfiles);

fromElem(elem, "linkswitches", linkswitches);

fromElem(elem, "libfiles", libfiles);

fromElem(elem, "libpaths", libpaths);

fromElem(elem, "deffile", deffile);

fromElem(elem, "resfile", resfile);

fromElem(elem, "exefile", exefile);

fromElem(elem, "useStdLibPath", useStdLibPath);

fromElem(elem, "cRuntime", cRuntime);

fromElem(elem, "privatePhobos", privatePhobos);

fromElem(elem, "additionalOptions", additionalOptions);

fromElem(elem, "preBuildCommand", preBuildCommand);

fromElem(elem, "postBuildCommand", postBuildCommand);

fromElem(elem, "debugtarget", debugtarget);

fromElem(elem, "debugarguments", debugarguments);

fromElem(elem, "debugworkingdir", debugworkingdir);

fromElem(elem, "debugattach", debugattach);

fromElem(elem, "debugremote", debugremote);

fromElem(elem, "debugEngine", debugEngine);

fromElem(elem, "debugStdOutToOutputWindow", debugStdOutToOutputWindow);

fromElem(elem, "pauseAfterRunning", pauseAfterRunning);

fromElem(elem, "filesToClean", filesToClean);

}

};

class ConfigProvider : DisposingComObject,

// IVsExtensibleObject,

IVsCfgProvider2,

IVsProjectCfgProvider

{

this(Project prj)

{

mProject = prj;

//                mConfigs ~= addref(new Config(this, "Debug"));

//                mConfigs ~= addref(new Config(this, "Release"));

}

Config addConfig(string name, string platform)

{

Config cfg = newCom!Config(this, name, platform);

mConfigs ~= addref(cfg);

return cfg;

}

void addConfigsToXml(xml.Document doc)

{

foreach(Config cfg; mConfigs)

{

auto config = new xml.Element("Config");

xml.setAttribute(config, "name", cfg.mName);

xml.setAttribute(config, "platform", cfg.mPlatform);

ProjectOptions opt = cfg.GetProjectOptions();

opt.writeXML(config);

doc ~= config;

}

}

override void Dispose()

{

foreach(Config cfg; mConfigs)

release(cfg);

mConfigs = mConfigs.init;

}

override HRESULT QueryInterface(in IID\* riid, void\*\* pvObject)

{

//mixin(LogCallMix);

if(queryInterface!(IVsCfgProvider) (this, riid, pvObject))

return S\_OK;

if(queryInterface!(IVsCfgProvider2) (this, riid, pvObject))

return S\_OK;

if(queryInterface!(IVsProjectCfgProvider) (this, riid, pvObject))

return S\_OK;

return super.QueryInterface(riid, pvObject);

}

// IVsCfgProvider

override int GetCfgs(

/\* [in] \*/ in ULONG celt,

/\* [size\_is][out][in] \*/ IVsCfg \*rgpcfg,

/\* [optional][out] \*/ ULONG \*pcActual,

/\* [optional][out] \*/ VSCFGFLAGS \*prgfFlags)

{

debug(FULL\_DBG) mixin(LogCallMix);

for(int i = 0; i < celt && i < mConfigs.length; i++)

rgpcfg[i] = addref(mConfigs[i]);

if(pcActual)

\*pcActual = mConfigs.length;

if(prgfFlags)

\*prgfFlags = cast(VSCFGFLAGS) 0;

return S\_OK;

}

// IVsProjectCfgProvider

override int OpenProjectCfg(

/\* [in] \*/ in wchar\* szProjectCfgCanonicalName,

/\* [out] \*/ IVsProjectCfg \*ppIVsProjectCfg)

{

mixin(LogCallMix);

return returnError(E\_NOTIMPL);

}

override int get\_UsesIndependentConfigurations(

/\* [out] \*/ BOOL \*pfUsesIndependentConfigurations)

{

logCall("%s.get\_UsesIndependentConfigurations(pfUsesIndependentConfigurations=%s)", this, \_toLog(pfUsesIndependentConfigurations));

return returnError(E\_NOTIMPL);

}

// IVsCfgProvider2

override int GetCfgNames(

/\* [in] \*/ in ULONG celt,

/\* [size\_is][out][in] \*/ BSTR \*rgbstr,

/\* [optional][out] \*/ ULONG \*pcActual)

{

mixin(LogCallMix);

int j, cnt = 0;

for(int i = 0; i < mConfigs.length; i++)

{

for(j = 0; j < i; j++)

if(mConfigs[i].mName == mConfigs[j].mName)

break;

if(j >= i)

{

if(cnt < celt && rgbstr)

rgbstr[cnt] = allocBSTR(mConfigs[i].mName);

cnt++;

}

}

if(pcActual)

\*pcActual = cnt;

return S\_OK;

}

override int GetPlatformNames(

/\* [in] \*/ in ULONG celt,

/\* [size\_is][out][in] \*/ BSTR \*rgbstr,

/\* [optional][out] \*/ ULONG \*pcActual)

{

mixin(LogCallMix);

int j, cnt = 0;

for(int i = 0; i < mConfigs.length; i++)

{

for(j = 0; j < i; j++)

if(mConfigs[i].mPlatform == mConfigs[j].mPlatform)

break;

if(j >= i)

{

if(cnt < celt)

rgbstr[cnt] = allocBSTR(mConfigs[i].mPlatform);

cnt++;

}

}

if(pcActual)

\*pcActual = cnt;

return S\_OK;

}

override int GetCfgOfName(

/\* [in] \*/ in wchar\* pszCfgName,

/\* [in] \*/ in wchar\* pszPlatformName,

/\* [out] \*/ IVsCfg \*ppCfg)

{

mixin(LogCallMix);

string cfg = to\_string(pszCfgName);

string plat = to\_string(pszPlatformName);

for(int i = 0; i < mConfigs.length; i++)

if((plat == "" || plat == mConfigs[i].mPlatform) &&

(cfg == "" || mConfigs[i].mName == cfg))

{

\*ppCfg = addref(mConfigs[i]);

return S\_OK;

}

return returnError(E\_INVALIDARG);

}

extern(D) void NotifyConfigEvent(void delegate(IVsCfgProviderEvents) dg)

{

// make a copy of the callback list, because it might change during execution of the callback

IVsCfgProviderEvents[] cbs;

foreach(cb; mCfgProviderEvents)

cbs ~= cb;

foreach(cb; cbs)

dg(cb);

}

override int AddCfgsOfCfgName(

/\* [in] \*/ in wchar\* pszCfgName,

/\* [in] \*/ in wchar\* pszCloneCfgName,

/\* [in] \*/ in BOOL fPrivate)

{

mixin(LogCallMix);

string strCfgName = to\_string(pszCfgName);

string strCloneCfgName = to\_string(pszCloneCfgName);

// Check if the CfgName already exists and that CloneCfgName exists

Config clonecfg;

foreach(c; mConfigs)

if(c.mName == strCfgName)

return returnError(E\_FAIL);

else if(c.mName == strCloneCfgName)

clonecfg = c;

if(strCloneCfgName.length && !clonecfg)

return returnError(E\_FAIL);

//if(!mProject.QueryEditProjectFile())

//        return returnError(E\_ABORT);

// copy configs for all platforms

int cnt = mConfigs.length;

for(int i = 0; i < cnt; i++)

if(mConfigs[i].mName == strCloneCfgName)

{

Config config = newCom!Config(this, strCfgName, mConfigs[i].mPlatform, mConfigs[i].mProjectOptions);

mConfigs ~= addref(config);

}

NotifyConfigEvent(delegate (IVsCfgProviderEvents cb) { cb.OnCfgNameAdded(pszCfgName); });

mProject.GetProjectNode().SetProjectFileDirty(true); // dirty the project file

return S\_OK;

}

override int DeleteCfgsOfCfgName(

/\* [in] \*/ in wchar\* pszCfgName)

{

logCall("%s.DeleteCfgsOfCfgName(pszCfgName=%s)", this, \_toLog(pszCfgName));

string strCfgName = to\_string(pszCfgName);

int cnt = mConfigs.length;

for(int i = 0; i < mConfigs.length; )

if(mConfigs[i].mName == strCfgName)

mConfigs = mConfigs[0..i] ~ mConfigs[i+1..$];

else

i++;

if(cnt == mConfigs.length)

return returnError(E\_FAIL);

NotifyConfigEvent(delegate (IVsCfgProviderEvents cb) { cb.OnCfgNameDeleted(pszCfgName); });

mProject.GetProjectNode().SetProjectFileDirty(true); // dirty the project file

return S\_OK;

}

override int RenameCfgsOfCfgName(

/\* [in] \*/ in wchar\* pszOldName,

/\* [in] \*/ in wchar\* pszNewName)

{

mixin(LogCallMix2);

string strOldName = to\_string(pszOldName);

string strNewName = to\_string(pszNewName);

Config config;

foreach(c; mConfigs)

if(c.mName == strNewName)

return returnError(E\_FAIL);

else if(c.mName == strOldName)

config = c;

if(!config)

return returnError(E\_FAIL);

//if(!mProject.QueryEditProjectFile())

//        return returnError(E\_ABORT);

foreach(c; mConfigs)

if(c.mName == strOldName)

c.mName = strNewName;

NotifyConfigEvent(delegate (IVsCfgProviderEvents cb) { cb.OnCfgNameRenamed(pszOldName, pszNewName); });

mProject.GetProjectNode().SetProjectFileDirty(true); // dirty the project file

return S\_OK;

}

override int AddCfgsOfPlatformName(

/\* [in] \*/ in wchar\* pszPlatformName,

/\* [in] \*/ in wchar\* pszClonePlatformName)

{

logCall("%s.AddCfgsOfPlatformName(pszPlatformName=%s,pszClonePlatformName=%s)", this, \_toLog(pszPlatformName), \_toLog(pszClonePlatformName));

string strPlatformName = to\_string(pszPlatformName);

string strClonePlatformName = to\_string(pszClonePlatformName);

// Check if the CfgName already exists and that CloneCfgName exists

Config clonecfg;

foreach(c; mConfigs)

if(c.mPlatform == strPlatformName)

return returnError(E\_FAIL);

else if(c.mPlatform == strClonePlatformName)

clonecfg = c;

if(strClonePlatformName.length && !clonecfg)

return returnError(E\_FAIL);

//if(!mProject.QueryEditProjectFile())

//        return returnError(E\_ABORT);

int cnt = mConfigs.length;

for(int i = 0; i < cnt; i++)

if(mConfigs[i].mPlatform == strClonePlatformName)

{

Config config = newCom!Config(this, mConfigs[i].mName, strPlatformName, mConfigs[i].mProjectOptions);

mConfigs ~= addref(config);

}

NotifyConfigEvent(delegate (IVsCfgProviderEvents cb) { cb.OnPlatformNameAdded(pszPlatformName); });

mProject.GetProjectNode().SetProjectFileDirty(true); // dirty the project file

return S\_OK;

}

override int DeleteCfgsOfPlatformName(

/\* [in] \*/ in wchar\* pszPlatformName)

{

logCall("%s.DeleteCfgsOfPlatformName(pszPlatformName=%s)", this, \_toLog(pszPlatformName));

string strPlatformName = to\_string(pszPlatformName);

int cnt = mConfigs.length;

for(int i = 0; i < mConfigs.length; )

if(mConfigs[i].mPlatform == strPlatformName)

mConfigs = mConfigs[0..i] ~ mConfigs[i+1..$];

else

i++;

if(cnt == mConfigs.length)

return returnError(E\_FAIL);

NotifyConfigEvent(delegate (IVsCfgProviderEvents cb) { cb.OnPlatformNameDeleted(pszPlatformName); });

mProject.GetProjectNode().SetProjectFileDirty(true); // dirty the project file

return S\_OK;

}

override int GetSupportedPlatformNames(

/\* [in] \*/ in ULONG celt,

/\* [size\_is][out][in] \*/ BSTR \*rgbstr,

/\* [optional][out] \*/ ULONG \*pcActual)

{

mixin(LogCallMix);

for(int cnt = 0; cnt < kPlatforms.length && cnt < celt && rgbstr; cnt++)

rgbstr[cnt] = allocBSTR(kPlatforms[cnt]);

if(pcActual)

\*pcActual = kPlatforms.length;

return S\_OK;

}

override int GetCfgProviderProperty(

/\* [in] \*/ in VSCFGPROPID propid,

/\* [out] \*/ VARIANT \*var)

{

mixin(LogCallMix);

switch(propid)

{

case VSCFGPROPID\_SupportsCfgAdd:

case VSCFGPROPID\_SupportsCfgDelete:

case VSCFGPROPID\_SupportsCfgRename:

case VSCFGPROPID\_SupportsPlatformAdd:

case VSCFGPROPID\_SupportsPlatformDelete:

var.vt = VT\_BOOL;

var.boolVal = true;

return S\_OK;

default:

break;

}

return returnError(E\_NOTIMPL);

}

override int AdviseCfgProviderEvents(

/\* [in] \*/ IVsCfgProviderEvents pCPE,

/\* [out] \*/ VSCOOKIE \*pdwCookie)

{

mixin(LogCallMix);

\*pdwCookie = ++mLastCfgProviderEventsCookie;

mCfgProviderEvents[mLastCfgProviderEventsCookie] = addref(pCPE);

return S\_OK;

}

override int UnadviseCfgProviderEvents(

/\* [in] \*/ in VSCOOKIE dwCookie)

{

logCall("%s.UnadviseCfgProviderEvents(dwCookie=%s)", this, \_toLog(dwCookie));

if(dwCookie in mCfgProviderEvents)

{

release(mCfgProviderEvents[dwCookie]);

mCfgProviderEvents.remove(dwCookie);

return S\_OK;

}

return returnError(E\_FAIL);

}

private:

Project mProject;

Config[] mConfigs;

IVsCfgProviderEvents[VSCOOKIE] mCfgProviderEvents;

VSCOOKIE mLastCfgProviderEventsCookie;

}

interface ConfigModifiedListener : IUnknown

{

void OnConfigModified();

}

class Config :        DisposingComObject,

IVsProjectCfg2,

IVsDebuggableProjectCfg,

IVsDebuggableProjectCfg2,

IVsBuildableProjectCfg,

IVsQueryDebuggableProjectCfg,

IVsProfilableProjectCfg,

ISpecifyPropertyPages

{

static const GUID iid = { 0x402744c1, 0xe382, 0x4877, [ 0x9e, 0x38, 0x26, 0x9c, 0xb7, 0xa3, 0xb8, 0x9d ] };

this(ConfigProvider provider, string name, string platform, ProjectOptions opts = null)

{

mProvider = provider;

if (opts)

{

mProjectOptions = clone(opts);

//mProjectOptions.setDebug(name == "Debug");

mProjectOptions.setX64(platform == "x64");

}

else

mProjectOptions = new ProjectOptions(name == "Debug", platform == "x64");

mBuilder = new CBuilderThread(this);

version(hasOutputGroup)

mOutputGroup = newCom!VsOutputGroup(this);

mName = name;

mPlatform = platform;

}

override void Dispose()

{

mBuilder.Dispose();

}

override ULONG AddRef()

{

return super.AddRef();

}

override ULONG Release()

{

return super.Release();

}

override HRESULT QueryInterface(in IID\* riid, void\*\* pvObject)

{

//mixin(LogCallMix);

if(queryInterface!(Config) (this, riid, pvObject))

return S\_OK;

if(queryInterface!(IVsCfg) (this, riid, pvObject))

return S\_OK;

if(queryInterface!(IVsProjectCfg) (this, riid, pvObject))

return S\_OK;

if(queryInterface!(IVsProjectCfg2) (this, riid, pvObject))

return S\_OK;

if(queryInterface!(ISpecifyPropertyPages) (this, riid, pvObject))

return S\_OK;

if(queryInterface!(IVsDebuggableProjectCfg) (this, riid, pvObject))

return S\_OK;

if(queryInterface!(IVsDebuggableProjectCfg2) (this, riid, pvObject))

return S\_OK;

if(queryInterface!(IVsBuildableProjectCfg) (this, riid, pvObject))

return S\_OK;

if(queryInterface!(IVsQueryDebuggableProjectCfg) (this, riid, pvObject))

return S\_OK;

version(hasProfilableConfig)

if(queryInterface!(IVsProfilableProjectCfg) (this, riid, pvObject))

return S\_OK;

return super.QueryInterface(riid, pvObject);

}

// ISpecifyPropertyPages

override int GetPages( /\* [out] \*/ CAUUID \*pPages)

{

mixin(LogCallMix);

CHierNode[] nodes;

CFileNode file;

CProjectNode proj;

if(GetProject().GetSelectedNodes(nodes) == S\_OK)

{

foreach(n; nodes)

{

if(!file)

file = cast(CFileNode) n;

if(!proj)

proj = cast(CProjectNode) n;

}

}

if (!proj)

return PropertyPageFactory.GetFilePages(pPages);

return PropertyPageFactory.GetProjectPages(pPages, false);

}

// IVsCfg

override int get\_DisplayName(BSTR \*pbstrDisplayName)

{

logCall("%s.get\_DisplayName(pbstrDisplayName=%s)", this, \_toLog(pbstrDisplayName));

\*pbstrDisplayName = allocBSTR(getCfgName());

return S\_OK;

}

override int get\_IsDebugOnly(BOOL \*pfIsDebugOnly)

{

logCall("%s.get\_IsDebugOnly(pfIsDebugOnly=%s)", this, \_toLog(pfIsDebugOnly));

\*pfIsDebugOnly = (mName == "Debug");

return S\_OK;

}

override int get\_IsReleaseOnly(BOOL \*pfIsReleaseOnly)

{

logCall("%s.get\_IsReleaseOnly(pfIsReleaseOnly=%s)", this, \_toLog(pfIsReleaseOnly));

\*pfIsReleaseOnly = (mName == "Release");

return S\_OK;

}

// IVsProjectCfg

override int EnumOutputs(IVsEnumOutputs \*ppIVsEnumOutputs)

{

mixin(LogCallMix);

\*ppIVsEnumOutputs = addref(newCom!DEnumOutputs(this, 0));

return S\_OK;

}

override int OpenOutput(in wchar\* szOutputCanonicalName, IVsOutput \*ppIVsOutput)

{

mixin(LogCallMix);

return returnError(E\_NOTIMPL);

}

override int get\_ProjectCfgProvider(/\* [out] \*/ IVsProjectCfgProvider \*ppIVsProjectCfgProvider)

{

mixin(LogCallMix);

\*ppIVsProjectCfgProvider = addref(mProvider);

return S\_OK;

}

override int get\_BuildableProjectCfg( /\* [out] \*/ IVsBuildableProjectCfg \*ppIVsBuildableProjectCfg)

{

mixin(LogCallMix);

\*ppIVsBuildableProjectCfg = addref(this);

return S\_OK;

}

override int get\_CanonicalName( /\* [out] \*/ BSTR \*pbstrCanonicalName)

{

logCall("get\_CanonicalName(pbstrCanonicalName=%s)", \_toLog(pbstrCanonicalName));

\*pbstrCanonicalName = allocBSTR(getName());

return S\_OK;

}

override int get\_Platform( /\* [out] \*/ GUID \*pguidPlatform)

{

// The documentation says this is obsolete, so don't do anything.

mixin(LogCallMix);

\*pguidPlatform = GUID(); //GUID\_VS\_PLATFORM\_WIN32\_X86;

return returnError(E\_NOTIMPL);

}

override int get\_IsPackaged( /\* [out] \*/ BOOL \*pfIsPackaged)

{

logCall("get\_IsPackaged(pfIsPackaged=%s)", \_toLog(pfIsPackaged));

return returnError(E\_NOTIMPL);

}

override int get\_IsSpecifyingOutputSupported( /\* [out] \*/ BOOL \*pfIsSpecifyingOutputSupported)

{

logCall("get\_IsSpecifyingOutputSupported(pfIsSpecifyingOutputSupported=%s)", \_toLog(pfIsSpecifyingOutputSupported));

return returnError(E\_NOTIMPL);

}

override int get\_TargetCodePage( /\* [out] \*/ UINT \*puiTargetCodePage)

{

logCall("get\_TargetCodePage(puiTargetCodePage=%s)", \_toLog(puiTargetCodePage));

return returnError(E\_NOTIMPL);

}

override int get\_UpdateSequenceNumber( /\* [out] \*/ ULARGE\_INTEGER \*puliUSN)

{

mixin(LogCallMix);

return returnError(E\_NOTIMPL);

}

override int get\_RootURL( /\* [out] \*/ BSTR \*pbstrRootURL)

{

logCall("get\_RootURL(pbstrRootURL=%s)", \_toLog(pbstrRootURL));

return returnError(E\_NOTIMPL);

}

// IVsProjectCfg2

override int get\_CfgType(

/\* [in] \*/ in IID\* iidCfg,

/\* [iid\_is][out] \*/ void \*\*ppCfg)

{

debug(FULL\_DBG) mixin(LogCallMix);

return QueryInterface(iidCfg, ppCfg);

}

override int get\_OutputGroups(

/\* [in] \*/ in ULONG celt,

/\* [size\_is][out][in] \*/ IVsOutputGroup \*rgpcfg,

/\* [optional][out] \*/ ULONG \*pcActual)

{

mixin(LogCallMix);

version(hasOutputGroup)

{

if(celt >= 1)

\*rgpcfg = addref(mOutputGroup);

if(pcActual)

\*pcActual = 1;

return S\_OK;

}

else

{

return returnError(E\_NOTIMPL);

}

}

override int OpenOutputGroup(

/\* [in] \*/ in wchar\* szCanonicalName,

/\* [out] \*/ IVsOutputGroup \*ppIVsOutputGroup)

{

mixin(LogCallMix);

version(hasOutputGroup)

{

if(to\_wstring(szCanonicalName) != to\_wstring(VS\_OUTPUTGROUP\_CNAME\_Built))

return returnError(E\_INVALIDARG);

\*ppIVsOutputGroup = addref(mOutputGroup);

return S\_OK;

}

else

{

return returnError(E\_NOTIMPL);

}

}

override int OutputsRequireAppRoot(

/\* [out] \*/ BOOL \*pfRequiresAppRoot)

{

logCall("%s.OutputsRequireAppRoot(pfRequiresAppRoot=%s)", this, \_toLog(pfRequiresAppRoot));

return returnError(E\_NOTIMPL);

}

override int get\_VirtualRoot(

/\* [out] \*/ BSTR \*pbstrVRoot)

{

logCall("%s.get\_VirtualRoot(pbstrVRoot=%s)", this, \_toLog(pbstrVRoot));

return returnError(E\_NOTIMPL);

}

override int get\_IsPrivate(

/\* [out] \*/ BOOL \*pfPrivate)

{

logCall("%s.get\_IsPrivate(pfPrivate=%s)", this, \_toLog(pfPrivate));

return returnError(E\_NOTIMPL);

}

// IVsDebuggableProjectCfg

override int DebugLaunch(

/\* [in] \*/ in VSDBGLAUNCHFLAGS grfLaunch)

{

logCall("%s.DebugLaunch(grfLaunch=%s)", this, \_toLog(grfLaunch));

string prg = mProjectOptions.replaceEnvironment(mProjectOptions.debugtarget, this);

if (prg.length == 0)

return S\_OK;

if(!isAbsolute(prg))

prg = GetProjectDir() ~ "\\" ~ prg;

//prg = quoteFilename(prg);

string workdir = mProjectOptions.replaceEnvironment(mProjectOptions.debugworkingdir, this);

if(!isAbsolute(workdir))

workdir = GetProjectDir() ~ "\\" ~ workdir;

Package.GetGlobalOptions().addExecutionPath(workdir);

string args = mProjectOptions.replaceEnvironment(mProjectOptions.debugarguments, this);

if(DBGLAUNCH\_NoDebug & grfLaunch)

{

if(mProjectOptions.pauseAfterRunning)

{

args = "/c " ~ quoteFilename(prg) ~ " " ~ args ~ " & pause";

prg = getCmdPath();

}

ShellExecuteW(null, null, toUTF16z(quoteFilename(prg)), toUTF16z(args), toUTF16z(workdir), SW\_SHOWNORMAL);

return(S\_OK);

}

return \_DebugLaunch(prg, workdir, args, mProjectOptions.debugEngine);

}

GUID getDebugEngineUID(int engine)

{

switch(engine)

{

case 1:

GUID GUID\_MaGoDebugger = uuid("{97348AC0-2B6B-4B99-A245-4C7E2C09D403}");

return GUID\_MaGoDebugger;

case 2:

return GUID\_COMPlusNativeEng; // the mixed-mode debugger (works only on x86)

default:

return GUID\_NativeOnlyEng; // works for x64

}

}

HRESULT \_DebugLaunch(string prg, string workdir, string args, int engine)

{

HRESULT hr = E\_NOTIMPL;

// When the debug target is the project build output, the project have to use

// IVsSolutionDebuggingAssistant2 to determine if the target was deployed.

// The interface allows the project to find out where the outputs were deployed to

// and direct the debugger to the deployed locations as appropriate.

// Projects start out their debugging sessions by calling MapOutputToDeployedURLs().

// Here we do not use IVsSolutionDebuggingAssistant2 because our debug target is

// explicitly set in the project options and it is not built by the project.

// For demo of how to use IVsSolutionDebuggingAssistant2 refer to MycPrj sample in the

// Environment SDK.

if(IVsDebugger srpVsDebugger = queryService!(IVsDebugger))

{

scope(exit) release(srpVsDebugger);

// if bstr-parameters not passed as BSTR parameters, VS2010 crashes on some systems

// not sure if they can be free'd afterwards...

VsDebugTargetInfo dbgi;

dbgi.cbSize = VsDebugTargetInfo.sizeof;

dbgi.bstrRemoteMachine = null;

string remote = mProjectOptions.replaceEnvironment(mProjectOptions.debugremote, this);

if(remote.length == 0)

{

if(!std.file.exists(prg))

{

UtilMessageBox("The program to launch does not exist:\n" ~ prg, MB\_OK, "Launch Debugger");

return S\_FALSE;

}

if(workdir.length && !isExistingDir(workdir))

{

UtilMessageBox("The working directory does not exist:\n" ~ workdir, MB\_OK, "Launch Debugger");

return S\_FALSE;

}

}

else

dbgi.bstrRemoteMachine = allocBSTR(remote); // \_toUTF16z(remote);

dbgi.dlo = DLO\_CreateProcess; // DLO\_Custom; // specifies how this process should be launched

// clsidCustom is the clsid of the debug engine to use to launch the debugger

dbgi.clsidCustom = getDebugEngineUID(engine);

dbgi.bstrMdmRegisteredName = null; // used with DLO\_AlreadyRunning. The name of the

// app as it is registered with the MDM.

dbgi.bstrExe = allocBSTR(prg); // \_toUTF16z(prg);

dbgi.bstrCurDir = allocBSTR(workdir); // \_toUTF16z(workdir);

dbgi.bstrArg = allocBSTR(args); // \_toUTF16z(args);

dbgi.fSendStdoutToOutputWindow = mProjectOptions.debugStdOutToOutputWindow;

hr = srpVsDebugger.LaunchDebugTargets(1, &dbgi);

if (FAILED(hr))

{

string msg = format("cannot launch debugger on %s\nhr = %x", prg, hr);

mProvider.mProject.SetErrorInfo(E\_FAIL, msg);

hr = E\_FAIL;

}

}

return(hr);

}

override int QueryDebugLaunch(

/\* [in] \*/ in VSDBGLAUNCHFLAGS grfLaunch,

/\* [out] \*/ BOOL \*pfCanLaunch)

{

//                mixin(LogCallMix);

\*pfCanLaunch = true;

return S\_OK; // returnError(E\_NOTIMPL);

}

// IVsDebuggableProjectCfg2

HRESULT OnBeforeDebugLaunch(in VSDBGLAUNCHFLAGS grfLaunch)

{

mixin(LogCallMix);

return S\_OK; // returnError(E\_NOTIMPL);

}

// IVsQueryDebuggableProjectCfg

HRESULT QueryDebugTargets(in VSDBGLAUNCHFLAGS grfLaunch, in ULONG cTargets,

VsDebugTargetInfo2 \*dti, ULONG \*pcActual)

{

if(cTargets > 0)

{

if(!dti)

return E\_INVALIDARG;

string remote = mProjectOptions.replaceEnvironment(mProjectOptions.debugremote, this);

string prg = mProjectOptions.replaceEnvironment(mProjectOptions.debugtarget, this);

string args = mProjectOptions.replaceEnvironment(mProjectOptions.debugarguments, this);

string workdir = mProjectOptions.replaceEnvironment(mProjectOptions.debugworkingdir, this);

if(!isAbsolute(workdir))

workdir = GetProjectDir() ~ "\\" ~ workdir;

prg = makeFilenameAbsolute(prg, workdir);

dti.cbSize = VsDebugTargetInfo2.sizeof;

dti.dlo = DLO\_CreateProcess; // specifies how this process should be launched or attached

dti.LaunchFlags = grfLaunch; // launch flags that were passed to IVsDebuggableProjectCfg::Launch

dti.bstrRemoteMachine = remote.length ? allocBSTR(remote) : null; // NULL for local machine, or remote machine name

dti.bstrExe = allocBSTR(prg);

dti.bstrArg = allocBSTR(args);

dti.bstrCurDir = allocBSTR(workdir);

dti.bstrEnv = null;

dti.guidLaunchDebugEngine = getDebugEngineUID(mProjectOptions.debugEngine);

dti.dwDebugEngineCount = 1;

dti.pDebugEngines = cast(GUID\*)CoTaskMemAlloc(GUID.sizeof);

\*(dti.pDebugEngines) = dti.guidLaunchDebugEngine;

/+

dti.guidPortSupplier; // port supplier guid

dti.bstrPortName; // name of port from above supplier (NULL is fine)

dti.bstrOptions; // custom options, specific to each guidLaunchDebugEngine (NULL is recommended)

dti.hStdInput; // for file redirection

dti.hStdOutput; // for file redirection

dti.hStdError; // for file redirection

dti.fSendToOutputWindow; // if TRUE, stdout and stderr will be routed to the output window

dti.dwProcessId; // process id (DLO\_AlreadyRunning)

dti.pUnknown; // interface pointer - usage depends on DEBUG\_LAUNCH\_OPERATION

dti.guidProcessLanguage; // Language of the hosting process. Used to preload EE's

+/

}

if (pcActual)

\*pcActual = 1;

return S\_OK;

}

///////////////////////////////////////////////////////////////

// IVsProfilableProjectCfg

override HRESULT get\_SuppressSignedAssemblyWarnings(/+[retval, out]+/VARIANT\_BOOL\* suppress)

{

mixin(LogCallMix);

\*suppress = FALSE;

return S\_OK;

}

override HRESULT get\_LegacyWebSupportRequired(/+[retval, out]+/VARIANT\_BOOL\* required)

{

mixin(LogCallMix);

\*required = FALSE;

return S\_OK;

}

HRESULT GetSupportedProfilingTasks(/+[out]+/ SAFEARRAY \*tasks)

{

mixin(LogCallMix);

BSTR task = allocBSTR("ClassicCPUSampling");

int index = 0;

SafeArrayPutElement(tasks, &index, &task);

return S\_OK;

}

HRESULT BeforeLaunch(in BSTR profilingTask)

{

mixin(LogCallMix);

return S\_OK;

}

HRESULT BeforeTargetsLaunched()

{

mixin(LogCallMix);

return S\_OK;

}

HRESULT LaunchProfiler()

{

mixin(LogCallMix);

version(hasProfilableConfig)

{

IVsProfilerLauncher launcher;

GUID svcid = uuid\_SVsProfilerLauncher;

GUID clsid = uuid\_IVsProfilerLauncher;

if (IServiceProvider sp = visuald.dpackage.Package.s\_instance.getServiceProvider())

sp.QueryService(&svcid, &clsid, cast(void\*\*)&launcher);

if (!launcher)

return E\_NOTIMPL;

auto infos = addref(newCom!EnumVsProfilerTargetInfos(this));

scope(exit) release(launcher);

scope(exit) release(infos);

HRESULT hr = launcher.LaunchProfiler(infos);

return hr;

}

else

return returnError(E\_NOTIMPL);

}

HRESULT QueryProfilerTargetInfoEnum(/+[out]+/ IEnumVsProfilerTargetInfos \*targetsEnum)

{

version(hasProfilableConfig)

{

mixin(LogCallMix);

\*targetsEnum = addref(newCom!EnumVsProfilerTargetInfos(this));

return S\_OK;

}

else

return returnError(E\_NOTIMPL);

}

HRESULT AllBrowserTargetsFinished()

{

mixin(LogCallMix);

return S\_OK;

}

HRESULT ProfilerAnalysisFinished()

{

mixin(LogCallMix);

return S\_OK;

}

///////////////////////////////////////////////////////////////

// IVsBuildableProjectCfg

override int get\_ProjectCfg(

/\* [out] \*/ IVsProjectCfg \*ppIVsProjectCfg)

{

mixin(LogCallMix);

return returnError(E\_NOTIMPL);

}

override int AdviseBuildStatusCallback(

/\* [in] \*/ IVsBuildStatusCallback pIVsBuildStatusCallback,

/\* [out] \*/ VSCOOKIE \*pdwCookie)

{

mixin(LogCallMix);

\*pdwCookie = ++mLastBuildStatusCookie;

mBuildStatusCallbacks[mLastBuildStatusCookie] = addref(pIVsBuildStatusCallback);

mTicking[mLastBuildStatusCookie] = false;

mStarted[mLastBuildStatusCookie] = false;

return S\_OK;

}

override int UnadviseBuildStatusCallback(

/\* [in] \*/ in VSCOOKIE dwCookie)

{

//                mixin(LogCallMix);

if(dwCookie in mBuildStatusCallbacks)

{

release(mBuildStatusCallbacks[dwCookie]);

mBuildStatusCallbacks.remove(dwCookie);

mTicking.remove(dwCookie);

mStarted.remove(dwCookie);

return S\_OK;

}

return returnError(E\_FAIL);

}

override int StartBuild(

/\* [in] \*/ IVsOutputWindowPane pIVsOutputWindowPane,

/\* [in] \*/ in DWORD dwOptions)

{

mixin(LogCallMix);

if(dwOptions & VS\_BUILDABLEPROJECTCFGOPTS\_REBUILD)

return mBuilder.Start(CBuilderThread.Operation.eRebuild, pIVsOutputWindowPane);

return mBuilder.Start(CBuilderThread.Operation.eBuild, pIVsOutputWindowPane);

}

override int StartClean(

/\* [in] \*/ IVsOutputWindowPane pIVsOutputWindowPane,

/\* [in] \*/ in DWORD dwOptions)

{

mixin(LogCallMix);

return mBuilder.Start(CBuilderThread.Operation.eClean, pIVsOutputWindowPane);

}

override int StartUpToDateCheck(

/\* [in] \*/ IVsOutputWindowPane pIVsOutputWindowPane,

/\* [in] \*/ in DWORD dwOptions)

{

mixin(LogCallMix);

HRESULT rc = mBuilder.Start(CBuilderThread.Operation.eCheckUpToDate, pIVsOutputWindowPane);

return rc == S\_OK ? S\_OK : E\_FAIL; // E\_FAIL used to indicate "not uptodate"

//return returnError(E\_NOTIMPL); //S\_OK;

}

override int QueryStatus(

/\* [out] \*/ BOOL \*pfBuildDone)

{

logCall("%s.QueryStatus(pfBuildDone=%s)", this, \_toLog(pfBuildDone));

mBuilder.QueryStatus(pfBuildDone);

return S\_OK;

}

override int Stop(

/\* [in] \*/ in BOOL fSync)

{

logCall("%s.Stop(fSync=%s)", this, \_toLog(fSync));

mBuilder.Stop(fSync);

return S\_OK;

}

override int Wait(

/\* [in] \*/ in DWORD dwMilliseconds,

/\* [in] \*/ in BOOL fTickWhenMessageQNotEmpty)

{

mixin(LogCallMix);

return returnError(E\_NOTIMPL);

}

override int QueryStartBuild(

/\* [in] \*/ in DWORD dwOptions,

/\* [optional][out] \*/ BOOL \*pfSupported,

/\* [optional][out] \*/ BOOL \*pfReady)

{

debug(FULL\_DBG) mixin(LogCallMix);

if(pfSupported)

\*pfSupported = true;

if(pfReady)

{

mBuilder.QueryStatus(pfReady);

}

return S\_OK; // returnError(E\_NOTIMPL);

}

override int QueryStartClean(

/\* [in] \*/ in DWORD dwOptions,

/\* [optional][out] \*/ BOOL \*pfSupported,

/\* [optional][out] \*/ BOOL \*pfReady)

{

mixin(LogCallMix);

if(pfSupported)

\*pfSupported = true;

if(pfReady)

{

mBuilder.QueryStatus(pfReady);

}

return S\_OK; // returnError(E\_NOTIMPL);

}

override int QueryStartUpToDateCheck(

/\* [in] \*/ in DWORD dwOptions,

/\* [optional][out] \*/ BOOL \*pfSupported,

/\* [optional][out] \*/ BOOL \*pfReady)

{

mixin(LogCallMix);

if(pfSupported)

\*pfSupported = true;

if(pfReady)

{

mBuilder.QueryStatus(pfReady);

}

return S\_OK; // returnError(E\_NOTIMPL);

}

//////////////////////////////////////////////////////////////////////////////

void AddModifiedListener(ConfigModifiedListener listener)

{

mModifiedListener.addunique(listener);

}

void RemoveModifiedListener(ConfigModifiedListener listener)

{

mModifiedListener.remove(listener);

}

//////////////////////////////////////////////////////////////////////////////

void SetDirty()

{

mProvider.mProject.GetProjectNode().SetProjectFileDirty(true);

foreach(listener; mModifiedListener)

listener.OnConfigModified();

}

CProjectNode GetProjectNode() { return mProvider.mProject.GetProjectNode(); }

string GetProjectPath() { return mProvider.mProject.GetFilename(); }

string GetProjectDir() { return dirName(mProvider.mProject.GetFilename()); }

string GetProjectName() { return mProvider.mProject.GetProjectNode().GetName(); }

Project GetProject() { return mProvider.mProject; }

ProjectOptions GetProjectOptions() { return mProjectOptions; }

string GetTargetPath()

{

string exe = mProjectOptions.getTargetPath();

return mProjectOptions.replaceEnvironment(exe, this);

}

string GetDependenciesPath()

{

string exe = mProjectOptions.getDependenciesPath();

return mProjectOptions.replaceEnvironment(exe, this);

}

string GetLinkDependenciesPath()

{

string dep = GetDependenciesPath();

assert(dep[$-4..$] == ".dep");

return dep[0..$-4] ~ ".lnkdep";

}

string GetCppCompiler()

{

switch(mProjectOptions.compiler)

{

default:

case Compiler.DMD: return mProjectOptions.mscoff || mProjectOptions.isX86\_64 ? "cl" : "dmc";

case Compiler.GDC: return "gcc";

case Compiler.LDC: return mProjectOptions.isLDCforMinGW() ? "clang" : "cl";

}

}

bool hasLinkDependencies()

{

return mProjectOptions.callLinkerDirectly() && Package.GetGlobalOptions().optlinkDeps;

}

string GetCommandLinePath()

{

string exe = mProjectOptions.getCommandLinePath();

return mProjectOptions.replaceEnvironment(exe, this);

}

string GetOutDir()

{

return mProjectOptions.replaceEnvironment(mProjectOptions.outdir, this);

}

string GetIntermediateDir()

{

return mProjectOptions.replaceEnvironment(mProjectOptions.objdir, this);

}

string[] GetDependencies(CFileNode file)

{

string tool = GetCompileTool(file);

if(tool == "Custom" || tool == kToolResourceCompiler || tool == kToolCpp)

{

string outfile = GetOutputFile(file);

string dep = file.GetDependencies(getCfgName());

dep = mProjectOptions.replaceEnvironment(dep, this, file.GetFilename(), outfile);

string[] deps = tokenizeArgs(dep);

deps ~= file.GetFilename();

string workdir = GetProjectDir();

foreach(ref string s; deps)

s = makeFilenameAbsolute(s, workdir);

return deps;

}

if(tool == "DMDsingle")

{

string outfile = GetOutputFile(file);

string depfile = outfile ~ ".dep";

depfile = mProjectOptions.replaceEnvironment(depfile, this, file.GetFilename(), outfile);

string workdir = GetProjectDir();

string deppath = makeFilenameAbsolute(depfile, workdir);

string[] files;

bool depok = false;

if(std.file.exists(deppath))

depok = getFilenamesFromDepFile(deppath, files);

if(!depok)

files ~= deppath; // force update without if dependency file does not exist or is invalid

files ~= file.GetFilename();

files ~= getDDocFileList();

makeFilenamesAbsolute(files, workdir);

return files;

}

return null;

}

bool isUptodate(CFileNode file, string\* preason)

{

string fcmd = GetCompileCommand(file);

if(fcmd.length == 0)

return true;

string outfile = GetOutputFile(file);

outfile = mProjectOptions.replaceEnvironment(outfile, this, file.GetFilename(), outfile);

string workdir = GetProjectDir();

string cmdfile = makeFilenameAbsolute(outfile ~ "." ~ kCmdLogFileExtension, workdir);

if(!compareCommandFile(cmdfile, fcmd))

{

if(preason)

\*preason = "command line has changed";

return false;

}

string[] deps = GetDependencies(file);

outfile = makeFilenameAbsolute(outfile, workdir);

string oldestFile, newestFile;

long targettm = getOldestFileTime( [ outfile ], oldestFile );

long sourcetm = getNewestFileTime(deps, newestFile);

if(targettm > sourcetm)

return true;

if(file.GetUptodateWithSameTime(getCfgName()) && targettm == sourcetm)

return true;

if(preason)

\*preason = newestFile ~ " is newer";

return false;

}

static bool IsResource(CFileNode file)

{

string tool = file.GetTool(null);

if(tool == "")

if(toLower(extension(file.GetFilename())) == ".rc")

return true;

return tool == kToolResourceCompiler;

}

static string GetStaticCompileTool(CFileNode file, string cfgname)

{

string tool = file.GetTool(cfgname);

if(tool == "")

{

string fname = file.GetFilename();

string ext = toLower(extension(fname));

if(isIn(ext, ".d", ".ddoc", ".def", ".lib", ".obj", ".o", ".res"))

tool = "DMD";

else if(ext == "rc")

tool = kToolResourceCompiler;

else if(isIn(ext, ".c", ".cpp", ".cxx", ".cc"))

tool = kToolCpp;

}

return tool;

}

string GetCompileTool(CFileNode file)

{

string tool = file.GetTool(getCfgName());

if(tool == "")

{

string fname = file.GetFilename();

string ext = toLower(extension(fname));

if(ext == ".d" && mProjectOptions.compilationModel == ProjectOptions.kSingleFileCompilation)

tool = "DMDsingle";

else if(isIn(ext, ".d", ".ddoc", ".def", ".lib", ".obj", ".o", ".res"))

tool = "DMD";

else if(ext == "rc")

tool = kToolResourceCompiler;

else if(isIn(ext, ".c", ".cpp", ".cxx", ".cc"))

tool = kToolCpp;

}

return tool;

}

string GetOutputFile(CFileNode file, string tool = null)

{

if(tool.empty)

tool = GetCompileTool(file);

string fname;

if(tool == "DMD")

return file.GetFilename();

if(tool == "DMDsingle")

fname = mProjectOptions.objdir ~ "\\" ~ safeFilename(stripExtension(file.GetFilename())) ~ "." ~ mProjectOptions.objectFileExtension();

if(tool == "RDMD")

fname = mProjectOptions.outdir ~ "\\" ~ safeFilename(stripExtension(file.GetFilename())) ~ ".exe";

if(tool == kToolResourceCompiler)

fname = mProjectOptions.objdir ~ "\\" ~ safeFilename(stripExtension(file.GetFilename()), "\_") ~ ".res";

if(tool == kToolCpp)

fname = mProjectOptions.objdir ~ "\\" ~ safeFilename(stripExtension(file.GetFilename()), "\_") ~ ".obj";

if(tool == "Custom")

fname = file.GetOutFile(getCfgName());

if(fname.length)

fname = mProjectOptions.replaceEnvironment(fname, this, file.GetFilename());

return fname;

}

string expandedAbsoluteFilename(string name)

{

string workdir = GetProjectDir();

string expname = mProjectOptions.replaceEnvironment(name, this);

string absname = makeFilenameAbsolute(expname, workdir);

return absname;

}

string GetBuildLogFile()

{

return expandedAbsoluteFilename("$(INTDIR)[\\$(SAFEPROJECTNAME).buildlog.html](file:///\\$(SAFEPROJECTNAME).buildlog.html)");

}

string[] GetBuildFiles()

{

string workdir = normalizeDir(GetProjectDir());

string outdir = normalizeDir(makeFilenameAbsolute(GetOutDir(), workdir));

string intermediatedir = normalizeDir(makeFilenameAbsolute(GetIntermediateDir(), workdir));

string target = makeFilenameAbsolute(GetTargetPath(), workdir);

string cmdfile = makeFilenameAbsolute(GetCommandLinePath(), workdir);

string[] files;

files ~= target;

files ~= cmdfile;

files ~= cmdfile ~ ".rsp";

files ~= makeFilenameAbsolute(GetDependenciesPath(), workdir);

files ~= makeFilenameAbsolute(GetLinkDependenciesPath(), workdir);

if(mProjectOptions.usesCv2pdb())

{

files ~= target ~ "\_cv";

files ~= setExtension(target, "pdb");

}

string mapfile = expandedAbsoluteFilename("$(INTDIR)[\\$(SAFEPROJECTNAME).map](file:///\\$(SAFEPROJECTNAME).map)");

files ~= mapfile;

string buildlog = GetBuildLogFile();

files ~= buildlog;

if(mProjectOptions.createImplib)

files ~= setExtension(target, "lib");

if(mProjectOptions.doDocComments)

{

if(mProjectOptions.docdir.length)

files ~= expandedAbsoluteFilename(normalizeDir(mProjectOptions.docdir)) ~ "\*.html";

if(mProjectOptions.docname.length)

files ~= expandedAbsoluteFilename(mProjectOptions.docname);

if(mProjectOptions.modules\_ddoc.length)

files ~= expandedAbsoluteFilename(mProjectOptions.modules\_ddoc);

}

if(mProjectOptions.doHdrGeneration)

{

if(mProjectOptions.hdrdir.length)

files ~= expandedAbsoluteFilename(normalizeDir(mProjectOptions.hdrdir)) ~ "\*.di";

if(mProjectOptions.hdrname.length)

files ~= expandedAbsoluteFilename(mProjectOptions.hdrname);

}

if(mProjectOptions.doXGeneration)

{

if(mProjectOptions.xfilename.length)

files ~= expandedAbsoluteFilename(mProjectOptions.xfilename);

}

string[] toclean = tokenizeArgs(mProjectOptions.filesToClean);

foreach(s; toclean)

{

string uqs = unquoteArgument(s);

files ~= outdir ~ uqs;

if(outdir != intermediatedir)

files ~= intermediatedir ~ uqs;

}

searchNode(mProvider.mProject.GetRootNode(),

delegate (CHierNode n) {

if(CFileNode file = cast(CFileNode) n)

{

string outname = GetOutputFile(file);

if (outname.length && outname != file.GetFilename())

{

files ~= makeFilenameAbsolute(outname, workdir);

files ~= makeFilenameAbsolute(outname ~ "." ~ kCmdLogFileExtension, workdir);

}

}

return false;

});

return files;

}

string GetCompileCommand(CFileNode file, bool syntaxOnly = false, string tool = null, string addopt = null)

{

if(tool.empty)

tool = GetCompileTool(file);

string cmd;

string outfile = GetOutputFile(file, tool);

if(tool == kToolResourceCompiler)

{

cmd = "rc /fo" ~ quoteFilename(outfile);

string include = Package.GetGlobalOptions().IncSearchPath;

if(include.length)

{

string[] incs = tokenizeArgs(include);

foreach(string inc; incs)

cmd ~= " /I" ~ quoteFilename(inc);

cmd = mProjectOptions.replaceEnvironment(cmd, this, outfile);

}

string addOpts = file.GetAdditionalOptions(getCfgName());

if(addOpts.length)

cmd ~= " " ~ addOpts;

cmd ~= " " ~ quoteFilename(file.GetFilename());

}

if(tool == kToolCpp)

{

cmd = mProjectOptions.getCppCommandLine(outfile, true);

string addOpts = file.GetAdditionalOptions(getCfgName());

if(addOpts.length)

cmd ~= " " ~ addOpts;

cmd ~= " " ~ quoteFilename(file.GetFilename());

}

if(tool == "Custom")

{

cmd = file.GetCustomCmd(getCfgName());

}

if(tool == "DMDsingle")

{

string depfile = GetOutputFile(file, tool) ~ ".dep";

cmd = "echo Compiling " ~ file.GetFilename() ~ "...\n";

cmd ~= mProjectOptions.buildCommandLine(true, false, false, syntaxOnly);

if(syntaxOnly && mProjectOptions.compiler == Compiler.GDC)

cmd ~= " -c -fsyntax-only";

else if(syntaxOnly)

cmd ~= " -c -o-";

else

cmd ~= " -c " ~ mProjectOptions.getOutputFileOption(outfile)

~ mProjectOptions.getDependenciesFileOption(depfile);

cmd ~= " " ~ file.GetFilename();

foreach(ddoc; getDDocFileList())

cmd ~= " " ~ ddoc;

}

if(tool == "RDMD")

{

// temporarily switch to "rdmd"

ProjectOptions opts = clone(mProjectOptions);

opts.compiler = Compiler.DMD;

opts.program = "rdmd";

opts.otherDMD = true;

opts.mapverbosity = 2; // no map option

opts.otherDMD = true;

opts.doXGeneration = false;

opts.doHdrGeneration = false;

opts.doDocComments = false;

opts.lib = OutputType.Executable;

//opts.runCv2pdb = false;

opts.exefile = "$(OutDir)\\" ~ baseName(stripExtension(outfile)) ~ ".exe";

cmd = "echo Compiling " ~ file.GetFilename() ~ "...\n";

cmd ~= opts.buildCommandLine(true, !syntaxOnly, false, syntaxOnly);

if(syntaxOnly)

cmd ~= " --build-only";

cmd ~= addopt ~ " " ~ file.GetFilename();

addopt = ""; // must be before filename for rdmd

if (!syntaxOnly)

{

string cv2pdb = opts.appendCv2pdb();

if (cv2pdb.length)

cmd ~= "\nif errorlevel 1 goto reportError\n" ~ opts.appendCv2pdb();

}

}

if(cmd.length)

{

cmd = getEnvironmentChanges() ~ cmd ~ addopt ~ "\n:reportError\n";

if(syntaxOnly)

cmd ~= "if errorlevel 1 echo Compiling " ~ file.GetFilename() ~ " failed!\n";

else

cmd ~= "if errorlevel 1 echo Building " ~ outfile ~ " failed!\n";

cmd = mProjectOptions.replaceEnvironment(cmd, this, file.GetFilename(), outfile);

}

return cmd;

}

string GetDisasmCommand(string objfile, string outfile)

{

bool x64 = mProjectOptions.isX86\_64;

bool mscoff = mProjectOptions.compiler == Compiler.DMD && mProjectOptions.mscoff;

GlobalOptions globOpt = Package.GetGlobalOptions();

string cmd = x64 ? mProjectOptions.compilerDirectories.DisasmCommand64 :

mscoff ? mProjectOptions.compilerDirectories.DisasmCommand32coff : mProjectOptions.compilerDirectories.DisasmCommand;

if(globOpt.demangleError)

{

string mangledfile = outfile ~ ".mangled";

cmd = mProjectOptions.replaceEnvironment(cmd, this, objfile, mangledfile);

cmd ~= "\nif errorlevel 0 \"" ~ Package.GetGlobalOptions().VisualDInstallDir ~ "dcxxfilt.exe\" < " ~ quoteFilename(mangledfile) ~ " > " ~ quoteFilename(outfile);

}

else

cmd = mProjectOptions.replaceEnvironment(cmd, this, objfile, outfile);

return cmd;

}

string getEnvironmentChanges()

{

string cmd;

bool x64 = mProjectOptions.isX86\_64;

bool mscoff = mProjectOptions.compiler == Compiler.DMD && mProjectOptions.mscoff;

GlobalOptions globOpt = Package.GetGlobalOptions();

string exeSearchPath = x64 ? mProjectOptions.compilerDirectories.ExeSearchPath64 :

mscoff ? mProjectOptions.compilerDirectories.ExeSearchPath32coff : mProjectOptions.compilerDirectories.ExeSearchPath;

if(exeSearchPath.length)

cmd ~= "set PATH=" ~ replaceCrLfSemi(exeSearchPath) ~ ";%PATH%\n";

string libSearchPath = x64 ? mProjectOptions.compilerDirectories.LibSearchPath64 :

mscoff ? mProjectOptions.compilerDirectories.LibSearchPath32coff : mProjectOptions.compilerDirectories.LibSearchPath;

bool hasGlobalPath = mProjectOptions.useStdLibPath && libSearchPath.length;

if(hasGlobalPath || mProjectOptions.libpaths.length)

{

// obsolete?

string lpath;

if(hasGlobalPath)

lpath = replaceCrLfSemi(libSearchPath);

if(mProjectOptions.libpaths.length && !\_endsWith(lpath, ";"))

lpath ~= ";";

lpath ~= mProjectOptions.libpaths;

if(mProjectOptions.compiler == Compiler.DMD)

cmd ~= "set DMD\_LIB=" ~ lpath ~ "\n";

else if(mProjectOptions.compiler == Compiler.LDC)

cmd ~= "set LIB=" ~ lpath ~ "\n";

}

if(mProjectOptions.useMSVCRT())

{

if(globOpt.WindowsSdkDir.length)

cmd ~= "set WindowsSdkDir=" ~ globOpt.WindowsSdkDir ~ "\n";

if(globOpt.VCInstallDir.length)

cmd ~= "set VCINSTALLDIR=" ~ globOpt.VCInstallDir ~ "\n";

}

return cmd;

}

string getModuleName(string fname)

{

string ext = toLower(extension(fname));

if(ext != ".d" && ext != ".di")

return "";

string modname = getModuleDeclarationName(fname);

if(modname.length > 0)

return modname;

return stripExtension(baseName(fname));

}

string getModulesDDocCommandLine(string[] files, ref string modules\_ddoc)

{

if(!mProjectOptions.doDocComments)

return "";

string mod\_cmd;

modules\_ddoc = strip(mProjectOptions.modules\_ddoc);

if(modules\_ddoc.length > 0)

{

modules\_ddoc = quoteFilename(modules\_ddoc);

mod\_cmd = "echo MODULES = >" ~ modules\_ddoc ~ "\n";

string workdir = GetProjectDir();

for(int i = 0; i < files.length; i++)

{

string fname = makeFilenameAbsolute(files[i], workdir);

string mod = getModuleName(fname);

if(mod.length > 0)

{

if(indexOf(mod, '.') < 0)

mod = "." ~ mod;

mod\_cmd ~= "echo $$(MODULE " ~ mod ~ ") >>" ~ modules\_ddoc ~ "\n";

}

}

}

return mod\_cmd;

}

string getCommandFileList(string[] files, string responsefile, ref string precmd)

{

if(mProjectOptions.compiler == Compiler.GDC)

foreach(ref f; files)

f = replace(f, "\\", "/");

files = files.dup;

quoteFilenames(files);

string fcmd = std.string.join(files, " ");

if(fcmd.length > 100)

{

precmd ~= "\n";

precmd ~= "echo " ~ files[0] ~ " >" ~ quoteFilename(responsefile) ~ "\n";

for(int i = 1; i < files.length; i++)

precmd ~= "echo " ~ files[i] ~ " >>" ~ quoteFilename(responsefile) ~ "\n";

precmd ~= "\n";

fcmd = " @" ~ quoteFilename(responsefile);

}

else if (fcmd.length)

fcmd = " " ~ fcmd;

if(mProjectOptions.compiler == Compiler.GDC && mProjectOptions.libfiles.length)

fcmd ~= " " ~ replace(mProjectOptions.libfiles, "\\", "/");

return fcmd;

}

string[] getObjectFileList(string[] dfiles)

{

string[] files = dfiles.dup;

string[] remove;

bool singleObj = (mProjectOptions.compilationModel == ProjectOptions.kCombinedCompileAndLink);

string targetObj;

foreach(ref f; files)

if(f.endsWith(".d") || f.endsWith(".D"))

{

if(singleObj)

{

if(targetObj.length)

remove ~= f;

else

{

targetObj = "$(OutDir)[\\$(ProjectName](file:///\\$(ProjectName))." ~ mProjectOptions.objectFileExtension();

f = targetObj;

}

}

else

{

string fname = stripExtension(f);

if(!mProjectOptions.preservePaths)

fname = baseName(fname);

fname ~= "." ~ mProjectOptions.objectFileExtension();

if(mProjectOptions.compiler.isIn(Compiler.DMD, Compiler.LDC) && !isAbsolute(fname))

f = mProjectOptions.objdir ~ "\\" ~ fname;

else

f = fname;

}

}

foreach(r; remove)

files.remove(r);

return files;

}

string getLinkFileList(string[] dfiles, ref string precmd)

{

string[] files = getObjectFileList(dfiles);

string responsefile = GetCommandLinePath() ~ ".lnkarg";

return getCommandFileList(files, responsefile, precmd);

}

string[] getSourceFileList()

{

string[] files;

searchNode(mProvider.mProject.GetRootNode(),

delegate (CHierNode n) {

if(CFileNode file = cast(CFileNode) n)

files ~= file.GetFilename();

return false;

});

return files;

}

string[] getDDocFileList()

{

string[] files;

searchNode(mProvider.mProject.GetRootNode(),

delegate (CHierNode n) {

if(CFileNode file = cast(CFileNode) n)

{

string fname = file.GetFilename();

if(extension(fname) == ".ddoc")

files ~= fname;

}

return false;

});

return files;

}

string[] getInputFileList()

{

string[] files;

searchNode(mProvider.mProject.GetRootNode(),

delegate (CHierNode n) {

if(CFileNode file = cast(CFileNode) n)

{

string fname = GetOutputFile(file);

if(fname.length)

if(file.GetTool(getCfgName()) != "Custom" || file.GetLinkOutput(getCfgName()))

files ~= fname;

}

return false;

});

string[] libs = getLibsFromDependentProjects();

foreach(lib; libs)

{

// dmd also understands ".json", ".map" and ".exe", but these are shortcuts for output files

string ext = toLower(extension(lib));

if(ext.isIn(".d", ".di", ".o", ".obj", ".lib", ".a", ".ddoc", ".res", ".def", ".dd", ".htm", ".html", ".xhtml"))

files ~= lib;

}

return files;

}

string GetPhobosPath()

{

string libpath = normalizeDir(GetIntermediateDir());

string libfile = "privatephobos.lib";

return libpath ~ libfile;

}

string GetPhobosCommandLine()

{

string libpath = normalizeDir(GetIntermediateDir());

bool x64 = mProjectOptions.isX86\_64;

bool mscoff = mProjectOptions.compiler == Compiler.DMD && mProjectOptions.mscoff;

string model = "32";

if(x64)

model = "64";

else if (mscoff)

model = "32mscoff";

string libfile = "privatephobos.lib";

string lib = libpath ~ libfile;

string cmdfile = libpath ~ "buildphobos.bat";

string dmddir = Package.GetGlobalOptions().findDmdBinDir();

string dmdpath = dmddir ~ "dmd.exe";

string installDir = normalizeDir(Package.GetGlobalOptions().DMD.InstallDir);

if(!std.file.exists(dmdpath))

return "echo dmd.exe not found in DMDInstallDir=" ~ installDir ~ " or through PATH\nexit /B 1";

string druntimePath = "src\\druntime\\src\\";

if(!std.file.exists(installDir ~ druntimePath ~ "object\_.d") &&

!std.file.exists(installDir ~ druntimePath ~ "object.d")) // dmd >=2.068 no longer has object\_.d

druntimePath = "druntime\\src\\";

if(!std.file.exists(installDir ~ druntimePath ~ "object\_.d") &&

!std.file.exists(installDir ~ druntimePath ~ "object.d"))

return "echo druntime source not found in DMDInstallDir=" ~ installDir ~ "\nexit /B 1";

string phobosPath = "src\\phobos\\";

if(!std.file.exists(installDir ~ phobosPath ~ "std"))

phobosPath = "phobos\\";

if(!std.file.exists(installDir ~ phobosPath ~ "std"))

return "echo phobos source not found in DMDInstallDir=" ~ installDir ~ "\nexit /B 1";

string cmdline = "@echo off\n";

cmdline ~= "echo Building " ~ lib ~ "\n";

cmdline ~= getEnvironmentChanges();

string opts = " -lib -d " ~ mProjectOptions.dmdCommonCompileOptions();

// collect C files

string[] cfiles;

cfiles ~= findDRuntimeFiles(installDir, druntimePath ~ "core", true, true, true);

cfiles ~= findDRuntimeFiles(installDir, phobosPath ~ "etc\\c", true, true, true);

if (cfiles.length)

{

foreach(i, ref file; cfiles)

{

file = installDir ~ file;

string outfile = libpath ~ "phobos-" ~ baseName(file) ~ ".obj";

string cccmd = mProjectOptions.getCppCommandLine(outfile, i == 0);

cmdline ~= cccmd ~ " -DNO\_snprintf " ~ file ~ "\n";

cmdline ~= "if errorlevel 1 exit /B %ERRORLEVEL%\n\n";

file = outfile;

}

}

// collect druntime D files

string[] files;

if(std.file.exists(installDir ~ druntimePath ~ "object\_.d"))

files ~= druntimePath ~ "object\_.d";

else

files ~= druntimePath ~ "object.d"; // dmd >=2.068 no longer has object.di

files ~= findDRuntimeFiles(installDir, druntimePath ~ "rt", true, false, true);

files ~= findDRuntimeFiles(installDir, druntimePath ~ "core", true, false, true);

files ~= findDRuntimeFiles(installDir, druntimePath ~ "gc", true, false, true);

foreach(ref file; files)

file = installDir ~ file;

files ~= cfiles;

if(model == "32")

files ~= installDir ~ druntimePath ~ "rt\\minit.obj";

string dmd;

if(mProjectOptions.otherDMD && mProjectOptions.program.length)

dmd = quoteNormalizeFilename(mProjectOptions.program);

else

dmd = "dmd";

static string buildFiles(string dmd, string outlib, string[] files)

{

string rspfile = outlib ~ ".rsp";

string qrspfile = quoteFilename(rspfile);

string cmdline = "echo. >" ~ qrspfile ~ "\n";

foreach(file; files)

cmdline ~= "echo " ~ quoteFilename(file) ~ " >>" ~ qrspfile ~ "\n";

cmdline ~= dmd ~ " -of" ~ quoteFilename(outlib) ~ " @" ~ qrspfile ~ "\n\n";

return cmdline;

}

// because of inconsistent object.di and object\_.d in dmd <2.067 we have to build

// druntime and phobos seperately

string druntimelib = libpath ~ "privatedruntime.lib";

cmdline ~= buildFiles(dmd ~ opts, druntimelib, files);

cmdline ~= "if errorlevel 1 exit /B %ERRORLEVEL%\n\n";

// collect phobos D files

files = null;

files ~= findDRuntimeFiles(installDir, phobosPath ~ "std", true, false, true);

files ~= findDRuntimeFiles(installDir, phobosPath ~ "etc\\c", true, false, true);

foreach(ref file; files)

file = installDir ~ file;

cmdline ~= buildFiles(dmd ~ opts ~ " " ~ quoteFilename(druntimelib), lib, files);

cmdline = mProjectOptions.replaceEnvironment(cmdline, this, null, lib);

return cmdline;

}

bool isPhobosUptodate(string\* preason)

{

string workdir = GetProjectDir();

string outfile = GetPhobosPath();

string lib = makeFilenameAbsolute(outfile, workdir);

if (!std.file.exists(lib))

{

if(preason)

\*preason = "does not exist";

return false;

}

string cmd = GetPhobosCommandLine();

if(cmd.length == 0)

return true;

string cmdfile = makeFilenameAbsolute(outfile ~ "." ~ kCmdLogFileExtension, workdir);

if(!compareCommandFile(cmdfile, cmd))

{

if(preason)

\*preason = "command line has changed";

return false;

}

// no further dependency checks

return true;

}

string getCommandLine()

{

bool doLink = mProjectOptions.compilationModel != ProjectOptions.kSeparateCompileOnly;

bool separateLink = mProjectOptions.doSeparateLink();

string opt = mProjectOptions.buildCommandLine(true, !separateLink && doLink, true);

string workdir = normalizeDir(GetProjectDir());

bool x64 = mProjectOptions.isX86\_64;

bool mscoff = mProjectOptions.compiler == Compiler.DMD && mProjectOptions.mscoff;

string precmd = getEnvironmentChanges();

string[] files = getInputFileList();

//quoteFilenames(files);

string responsefile = GetCommandLinePath() ~ ".rsp";

string fcmd = getCommandFileList(files, responsefile, precmd);

string[] srcfiles = getSourceFileList();

string modules\_ddoc;

string mod\_cmd = getModulesDDocCommandLine(srcfiles, modules\_ddoc);

if(mod\_cmd.length > 0)

{

precmd ~= mod\_cmd ~ "\nif errorlevel 1 goto reportError\n";

fcmd ~= " " ~ modules\_ddoc;

}

auto globOpts = Package.GetGlobalOptions();

if(separateLink || !doLink)

{

bool singleObj = (mProjectOptions.compilationModel == ProjectOptions.kCombinedCompileAndLink);

if(fcmd.length == 0)

opt = ""; // don't try to build zero files

else if(singleObj)

opt ~= " -c" ~ mProjectOptions.getOutputFileOption("$(OutDir)\\$(ProjectName)." ~ mProjectOptions.objectFileExtension());

else

opt ~= " -c" ~ mProjectOptions.getOutputDirOption();

}

string addopt;

if(mProjectOptions.additionalOptions.length && fcmd.length)

addopt = " " ~ mProjectOptions.additionalOptions.replace("\n", " ");

string cmd = precmd ~ opt ~ fcmd ~ addopt ~ "\n";

cmd = cmd ~ "if errorlevel 1 goto reportError\n";

if(separateLink && doLink)

{

string prelnk, lnkcmd;

if(mProjectOptions.callLinkerDirectly())

{

string libpaths, options;

string otherCompiler = mProjectOptions.replaceEnvironment(mProjectOptions.otherCompilerPath(), this);

string linkpath = globOpts.getLinkerPath(x64, mscoff, workdir, otherCompiler, &libpaths, &options);

lnkcmd = quoteFilename(linkpath) ~ " ";

if(globOpts.demangleError || globOpts.optlinkDeps)

lnkcmd = "\"$(VisualDInstallDir)pipedmd.exe\" "

~ (globOpts.demangleError ? null : "-nodemangle ")

~ (globOpts.optlinkDeps ? "-deps " ~ quoteFilename(GetLinkDependenciesPath()) ~ " " : null)

~ lnkcmd;

string[] lnkfiles = getObjectFileList(files); // convert D files to object files, but leaves anything else untouched

string cmdfiles = mProjectOptions.optlinkCommandLine(lnkfiles, options, workdir, x64 || mscoff);

if(cmdfiles.length > 100)

{

string lnkresponsefile = GetCommandLinePath() ~ ".lnkarg";

lnkresponsefile = makeFilenameAbsolute(lnkresponsefile, workdir);

if(lnkresponsefile != quoteFilename(lnkresponsefile))

{

// optlink does not support quoted response files

if(!std.file.exists(lnkresponsefile))

collectException(std.file.write(lnkresponsefile, ""));

string shortresponsefile = shortFilename(lnkresponsefile);

if (shortresponsefile.empty || shortresponsefile != quoteFilename(shortresponsefile))

lnkresponsefile = baseName(lnkresponsefile); // if short name generation fails, move it into the project folder

else

lnkresponsefile = shortresponsefile;

}

prelnk ~= "echo. > " ~ lnkresponsefile ~ "\n";

prelnk ~= "echo " ~ cmdfiles.replace("+", "+ >> " ~ lnkresponsefile ~ "\necho ");

prelnk ~= " >> " ~ lnkresponsefile ~ "\n\n";

lnkcmd ~= "@" ~ lnkresponsefile;

}

else

lnkcmd ~= cmdfiles;

if(!mProjectOptions.useStdLibPath)

prelnk = "set OPTLINKS=%OPTLINKS% /NOSCANLIB\n" ~ prelnk;

prelnk = "set LIB=" ~ libpaths ~ "\n" ~ prelnk;

}

else

{

lnkcmd = mProjectOptions.buildCommandLine(false, true, false);

lnkcmd ~= getLinkFileList(files, prelnk);

string addlnkopt = mProjectOptions.getAdditionalLinkOptions();

if(addlnkopt.length)

lnkcmd ~= " " ~ addlnkopt;

}

cmd = cmd ~ "\n" ~ prelnk ~ lnkcmd ~ "\n";

cmd = cmd ~ "if errorlevel 1 goto reportError\n";

}

string cv2pdb = mProjectOptions.appendCv2pdb();

if(cv2pdb.length && doLink)

{

string cvtarget = quoteFilename(mProjectOptions.getTargetPath() ~ "\_cv");

cmd ~= "if not exist " ~ cvtarget ~ " (echo " ~ cvtarget ~ " not created! && goto reportError)\n";

cmd ~= "echo Converting debug information...\n";

cmd ~= cv2pdb;

cmd ~= "\nif errorlevel 1 goto reportError\n";

}

string pre = strip(mProjectOptions.preBuildCommand);

if(pre.length)

cmd = pre ~ "\nif errorlevel 1 goto reportError\n" ~ cmd;

string post = strip(mProjectOptions.postBuildCommand);

if(post.length)

cmd = cmd ~ "\nif errorlevel 1 goto reportError\n" ~ post ~ "\n\n";

string target = quoteFilename(mProjectOptions.getTargetPath());

cmd ~= "if not exist " ~ target ~ " (echo " ~ target ~ " not created! && goto reportError)\n";

cmd ~= "\ngoto noError\n";

cmd ~= "\n:reportError\n";

cmd ~= "echo Building " ~ GetTargetPath() ~ " failed!\n";

cmd ~= "\n:noError\n";

return mProjectOptions.replaceEnvironment(cmd, this);

}

bool writeLinkDependencyFile()

{

string workdir = normalizeDir(GetProjectDir());

string depfile = makeFilenameAbsolute(GetDependenciesPath(), workdir);

string[] files = getInputFileList();

files = getObjectFileList(files);

string prefix = "target (";

string postfix = ") : public : object \n";

string deps;

foreach(f; files)

{

deps ~= prefix ~ replace(f, "\\", "\\\\") ~ postfix;

}

bool fromMap = mProjectOptions.mapverbosity >= 3;

try

{

std.file.write(depfile, deps);

return true;

}

catch(Exception e)

{

}

return false;

}

string[] getLibsFromDependentProjects()

{

string[] libs;

auto solutionBuildManager = queryService!(IVsSolutionBuildManager)();

if(!solutionBuildManager)

return libs;

scope(exit) release(solutionBuildManager);

ULONG cActual;

if(HRESULT hr = solutionBuildManager.GetProjectDependencies(mProvider.mProject, 0, null, &cActual))

return libs;

IVsHierarchy[] pHier = new IVsHierarchy [cActual];

if(HRESULT hr = solutionBuildManager.GetProjectDependencies(mProvider.mProject, cActual, pHier.ptr, &cActual))

return libs;

for(int i = 0; i < cActual; i++)

{

IVsProjectCfg prjcfg;

if(pHier[i].QueryInterface(&IVsProjectCfg.iid, cast(void\*\*)&prjcfg) != S\_OK)

{

IVsCfg cfg;

IVsGetCfgProvider gcp;

IVsCfgProvider cp;

IVsCfgProvider2 cp2;

if(pHier[i].QueryInterface(&IVsGetCfgProvider.iid, cast(void\*\*)&gcp) == S\_OK)

gcp.GetCfgProvider(&cp);

else

pHier[i].QueryInterface(&IVsCfgProvider.iid, cast(void\*\*)&cp);

if(cp)

{

cp.QueryInterface(&IVsCfgProvider2.iid, cast(void\*\*)&cp2);

if(cp2)

{

cp2.GetCfgOfName(\_toUTF16z(mName), \_toUTF16z(mPlatform), &cfg);

if(!cfg)

cp2.GetCfgs(1, &cfg, null, null); // TODO: find a "similar" config?

if(cfg)

cfg.QueryInterface(&IVsProjectCfg.iid, cast(void\*\*)&prjcfg);

}

}

release(cfg);

release(gcp);

release(cp);

release(cp2);

}

if(prjcfg)

{

scope(exit) release(prjcfg);

debug logOutputGroups(prjcfg);

version(none)

if(auto prjcfg2 = qi\_cast!IVsProjectCfg2(prjcfg))

{

scope(exit) release(prjcfg2);

IVsOutputGroup outputGroup;

if(prjcfg2.OpenOutputGroup(VS\_OUTPUTGROUP\_CNAME\_Built, &outputGroup) == S\_OK)

{

scope(exit) release(outputGroup);

ULONG cnt;

if(outputGroup.get\_Outputs(0, null, &cnt) == S\_OK)

{

auto outs = new IVsOutput2[cnt];

if(outputGroup.get\_Outputs(cnt, outs.ptr, null) == S\_OK)

{

foreach(o; outs)

{

ScopedBSTR target;

if(o.get\_CanonicalName(&target.bstr) == S\_OK)

{

string targ = target.detach();

libs ~= targ;

}

release(o);

}

}

}

}

}

IVsEnumOutputs eo;

if(prjcfg.EnumOutputs(&eo) == S\_OK)

{

scope(exit) release(eo);

ULONG fetched;

string lastTarg;

IVsOutput pIVsOutput;

while(eo.Next(1, &pIVsOutput, &fetched) == S\_OK && fetched == 1)

{

ScopedBSTR target;

if(pIVsOutput.get\_CanonicalName(&target.bstr) == S\_OK)

//if(pIVsOutput.get\_DeploySourceURL(&target.bstr) == S\_OK)

//if(pIVsOutput.get\_DisplayName(&target.bstr) == S\_OK)

{

string targ = target.detach();

if (lastTarg.length && targ.indexOf('$') >= 0)

{

// VC projects report the import library without expanding macros

// (even if building static libraries), so assume it lies along side the DLL

if (targ.extension().toLower() == ".lib" && lastTarg.extension().toLower() != ".lib")

targ = lastTarg.stripExtension() ~ ".lib";

else

targ = null;

}

if (targ.length)

{

libs ~= targ;

lastTarg = targ;

}

}

release(pIVsOutput);

}

}

}

release(pHier[i]);

}

return libs;

}

void logOutputGroups(IVsProjectCfg prjcfg)

{

if(auto prjcfg2 = qi\_cast!IVsProjectCfg2(prjcfg))

{

scope(exit) release(prjcfg2);

ULONG cntGroups;

if(SUCCEEDED(prjcfg2.get\_OutputGroups(0, null, &cntGroups)))

{

auto groups = new IVsOutputGroup[cntGroups];

if(prjcfg2.get\_OutputGroups(cntGroups, groups.ptr, &cntGroups) == S\_OK)

{

foreach(outputGroup; groups)

{

scope(exit) release(outputGroup);

BSTR bstrCanName, bstrDispName, bstrKeyOut, bstrDesc;

outputGroup.get\_CanonicalName(&bstrCanName);

outputGroup.get\_DisplayName(&bstrDispName);

outputGroup.get\_KeyOutput(&bstrKeyOut);

outputGroup.get\_Description(&bstrDesc);

logCall("Group: %s Disp: %s KeyOut: %s Desc: %s", detachBSTR(bstrCanName), detachBSTR(bstrDispName), detachBSTR(bstrKeyOut), detachBSTR(bstrDesc));

ULONG cnt;

if(outputGroup.get\_Outputs(0, null, &cnt) == S\_OK)

{

auto outs = new IVsOutput2[cnt];

if(outputGroup.get\_Outputs(cnt, outs.ptr, &cnt) == S\_OK)

{

foreach(o; outs)

{

BSTR target, display, url;

o.get\_CanonicalName(&target);

o.get\_DisplayName(&display);

o.get\_DeploySourceURL(&url);

logCall(" Out: %s Disp: %s URL: %s", detachBSTR(target), detachBSTR(display), detachBSTR(url));

release(o);

}

}

}

}

}

}

}

}

int addJSONFiles(ref string[] files)

{

int cnt = 0;

alias mProjectOptions opt;

if(opt.doXGeneration)

{

void addJSONFile(string xfile)

{

xfile = makeFilenameAbsolute(xfile, GetProjectDir());

if(xfile.length && std.file.exists(xfile))

{

addunique(files, xfile);

cnt++;

}

}

if(opt.compilationModel == ProjectOptions.kSingleFileCompilation)

{

searchNode(mProvider.mProject.GetRootNode(),

delegate (CHierNode n) {

if(CFileNode file = cast(CFileNode) n)

{

string tool = GetCompileTool(file);

if(tool == "DMDsingle")

{

string outfile = GetOutputFile(file);

string xfile = opt.replaceEnvironment(opt.xfilename, this, file.GetFilename(), outfile);

addJSONFile(xfile);

}

}

return false;

});

}

else

{

string xfile = opt.replaceEnvironment(opt.xfilename, this);

addJSONFile(xfile);

}

}

return cnt;

}

// tick the sink and check if build can continue or not.

BOOL FFireTick()

{

foreach(cb; mBuildStatusCallbacks)

{

//if (m\_rgfTicking[i])

{

BOOL fContinue = TRUE;

HRESULT hr = cb.Tick(&fContinue);

assert(SUCCEEDED(hr));

if (!fContinue)

return FALSE;

}

}

return TRUE;

}

void FFireBuildBegin(ref BOOL fContinue)

{

fContinue = TRUE;

foreach(key, cb; mBuildStatusCallbacks)

{

HRESULT hr = cb.BuildBegin(&fContinue);

if(FAILED(hr) || !fContinue)

break;

mStarted[key] = true;

}

}

void FFireBuildEnd(BOOL fSuccess)

{

// make a copy in case BuildEnd calls Unadvise

IVsBuildStatusCallback[] cbs;

foreach(key, cb; mBuildStatusCallbacks)

if(mStarted[key])

{

cbs ~= cb;

mStarted[key] = false;

}

foreach(cb; cbs)

{

HRESULT hr = cb.BuildEnd(fSuccess);

assert(SUCCEEDED(hr));

}

Package.scheduleUpdateLibrary();

}

CBuilderThread getBuilder() { return mBuilder; }

string getName() { return mName; }

string getPlatform() { return mPlatform; }

string getCfgName() { return mName ~ "|" ~ mPlatform; }

private:

string mName;

string mPlatform;

ConfigProvider mProvider;

ProjectOptions mProjectOptions;

CBuilderThread mBuilder;

version(hasOutputGroup)

VsOutputGroup mOutputGroup;

ConfigModifiedListener[] mModifiedListener;

IVsBuildStatusCallback[VSCOOKIE] mBuildStatusCallbacks;

bool[VSCOOKIE] mTicking;

bool[VSCOOKIE] mStarted;

VSCOOKIE mLastBuildStatusCookie;

};

class DEnumOutFactory : DComObject, IClassFactory

{

override HRESULT QueryInterface(in IID\* riid, void\*\* pvObject)

{

if(queryInterface2!(IClassFactory) (this, IID\_IClassFactory, riid, pvObject))

return S\_OK;

return super.QueryInterface(riid, pvObject);

}

override HRESULT CreateInstance(IUnknown UnkOuter, in IID\* riid, void\*\* pvObject)

{

logCall("%s.CreateInstance(riid=%s)", this, \_toLog(riid));

assert(!UnkOuter);

DEnumOutputs eo = newCom!DEnumOutputs(null, 0);

return eo.QueryInterface(riid, pvObject);

}

override HRESULT LockServer(in BOOL fLock)

{

return returnError(E\_NOTIMPL);

}

}

class DEnumOutputs : DComObject, IVsEnumOutputs, ICallFactory, IExternalConnection, IMarshal

{

// {785486EE-2FB9-47f5-85A9-5790A60B5CEB}

static const GUID iid = { 0x785486ee, 0x2fb9, 0x47f5, [ 0x85, 0xa9, 0x57, 0x90, 0xa6, 0xb, 0x5c, 0xeb ] };

string[] mTargets;

int mPos;

this(Config cfg, int pos)

{

if(cfg)

mTargets ~= makeFilenameAbsolute(cfg.GetTargetPath(), cfg.GetProjectDir());

mPos = pos;

}

this(DEnumOutputs eo)

{

mTargets = eo.mTargets;

mPos = eo.mPos;

}

override HRESULT QueryInterface(in IID\* riid, void\*\* pvObject)

{

if(queryInterface!(IVsEnumOutputs) (this, riid, pvObject))

return S\_OK;

if(queryInterface!(ICallFactory) (this, riid, pvObject))

return S\_OK;

if(queryInterface!(IExternalConnection) (this, riid, pvObject))

return S\_OK;

if(queryInterface!(IMarshal) (this, riid, pvObject))

return S\_OK;

return super.QueryInterface(riid, pvObject);

}

override HRESULT Reset()

{

mixin(LogCallMix);

mPos = 0;

return S\_OK;

}

override HRESULT Next(in ULONG cElements, IVsOutput \*rgpIVsOutput, ULONG \*pcElementsFetched)

{

mixin(LogCallMix);

if(mPos >= mTargets.length || cElements < 1)

{

if(pcElementsFetched)

\*pcElementsFetched = 0;

return returnError(S\_FALSE);

}

if(pcElementsFetched)

\*pcElementsFetched = 1;

\*rgpIVsOutput = addref(newCom!VsOutput(mTargets[mPos]));

mPos++;

return S\_OK;

}

override HRESULT Skip(in ULONG cElements)

{

logCall("%s.Skip(cElements=%s)", this, \_toLog(cElements));

mPos += cElements;

if(mPos > mTargets.length)

{

mPos = mTargets.length;

return S\_FALSE;

}

return S\_OK;

}

override HRESULT Clone(IVsEnumOutputs \*ppIVsEnumOutputs)

{

mixin(LogCallMix);

\*ppIVsEnumOutputs = addref(newCom!DEnumOutputs(this));

return S\_OK;

}

// ICallFactory

override HRESULT CreateCall(

/+[in]+/ in IID\* riid,

/+[in]+/ IUnknown pCtrlUnk,

/+[in]+/ in IID\* riid2,

/+[out, iid\_is(riid2)]+/ IUnknown \*ppv )

{

mixin(LogCallMix);

return returnError(E\_NOTIMPL);

}

// IExternalConnection

override DWORD AddConnection (

/+[in]+/ in DWORD extconn,

/+[in]+/ in DWORD reserved )

{

mixin(LogCallMix);

return ++mExternalReferences;

}

override DWORD ReleaseConnection(

/+[in]+/ in DWORD extconn,

/+[in]+/ in DWORD reserved,

/+[in]+/ in BOOL fLastReleaseCloses )

{

mixin(LogCallMix);

--mExternalReferences;

if(mExternalReferences == 0)

CoDisconnectObject(this, 0);

return mExternalReferences;

}

int mExternalReferences;

// IMarshall

override HRESULT GetUnmarshalClass

(

/+[in]+/ in IID\* riid,

/+[in, unique]+/ in void \*pv,

/+[in]+/ in DWORD dwDestContext,

/+[in, unique]+/ in void \*pvDestContext,

/+[in]+/ in DWORD mshlflags,

/+[out]+/ CLSID \*pCid

)

{

mixin(LogCallMixNoRet);

\*cast(GUID\*)pCid = g\_unmarshalEnumOutCLSID;

return S\_OK;

}

override HRESULT GetMarshalSizeMax

(

/+[in]+/ in IID\* riid,

/+[in, unique]+/ in void \*pv,

/+[in]+/ in DWORD dwDestContext,

/+[in, unique]+/ in void \*pvDestContext,

/+[in]+/ in DWORD mshlflags,

/+[out]+/ DWORD \*pSize

)

{

mixin(LogCallMixNoRet);

DWORD size = iid.sizeof + int.sizeof;

foreach(s; mTargets)

size += int.sizeof + s.length;

size += mPos.sizeof;

\*pSize = size;

return S\_OK;

//return returnError(E\_NOTIMPL);

}

override HRESULT MarshalInterface

(

/+[in, unique]+/ IStream pStm,

/+[in]+/ in IID\* riid,

/+[in, unique]+/ in void \*pv,

/+[in]+/ in DWORD dwDestContext,

/+[in, unique]+/ in void \*pvDestContext,

/+[in]+/ in DWORD mshlflags

)

{

mixin(LogCallMixNoRet);

if(HRESULT hr = pStm.Write(cast(void\*)&iid, iid.sizeof, null))

return hr;

int length = mTargets.length;

if(HRESULT hr = pStm.Write(&length, length.sizeof, null))

return hr;

foreach(s; mTargets)

{

length = s.length;

if(HRESULT hr = pStm.Write(&length, length.sizeof, null))

return hr;

if(HRESULT hr = pStm.Write(cast(void\*)s.ptr, length, null))

return hr;

}

if(HRESULT hr = pStm.Write(&mPos, mPos.sizeof, null))

return hr;

return S\_OK;

}

override HRESULT UnmarshalInterface

(

/+[in, unique]+/ IStream pStm,

/+[in]+/ in IID\* riid,

/+[out]+/ void \*\*ppv

)

{

mixin(LogCallMix);

GUID miid;

if(HRESULT hr = pStm.Read(&miid, iid.sizeof, null))

return returnError(hr);

assert(miid == iid);

int cnt;

if(HRESULT hr = pStm.Read(&cnt, cnt.sizeof, null))

return hr;

DEnumOutputs eo = newCom!DEnumOutputs(null, 0);

for(int i = 0; i < cnt; i++)

{

int length;

if(HRESULT hr = pStm.Read(&length, length.sizeof, null))

return hr;

char[] s = new char[length];

if(HRESULT hr = pStm.Read(s.ptr, length, null))

return hr;

eo.mTargets ~= cast(string) s;

}

if(HRESULT hr = pStm.Read(&eo.mPos, eo.mPos.sizeof, null))

return hr;

return eo.QueryInterface(riid, ppv);

}

override HRESULT ReleaseMarshalData(/+[in, unique]+/ IStream pStm)

{

mixin(LogCallMix2);

return returnError(E\_NOTIMPL);

}

override HRESULT DisconnectObject(/+[in]+/ in DWORD dwReserved)

{

logCall("%s.DisconnectObject(dwReserved=%s)", this, \_toLog(dwReserved));

return returnError(E\_NOTIMPL);

}

}

class VsOutput : DComObject, IVsOutput2

{

string mTarget;

this(string target)

{

mTarget = target;

}

override HRESULT QueryInterface(in IID\* riid, void\*\* pvObject)

{

version(hasOutputGroup)

if(queryInterface!(IVsOutput2) (this, riid, pvObject))

return S\_OK;

if(queryInterface!(IVsOutput) (this, riid, pvObject))

return S\_OK;

return super.QueryInterface(riid, pvObject);

}

override HRESULT get\_DisplayName(BSTR \*pbstrDisplayName)

{

logCall("%s.get\_DisplayName(pbstrDisplayName=%s)", this, \_toLog(pbstrDisplayName));

\*pbstrDisplayName = allocBSTR(mTarget);

return S\_OK;

}

override HRESULT get\_CanonicalName(BSTR \*pbstrCanonicalName)

{

logCall("%s.get\_CanonicalName(pbstrCanonicalName=%s)", this, \_toLog(pbstrCanonicalName));

\*pbstrCanonicalName = allocBSTR(mTarget);

return S\_OK;

}

override HRESULT get\_DeploySourceURL(BSTR \*pbstrDeploySourceURL)

{

logCall("%s.get\_DeploySourceURL(pbstrDeploySourceURL=%s)", this, \_toLog(pbstrDeploySourceURL));

\*pbstrDeploySourceURL = allocBSTR("[file:///](NULL)" ~ mTarget);

return S\_OK;

}

// obsolete method

override HRESULT get\_Type(/+[out]+/ GUID \*pguidType)

{

logCall("%s.get\_Type(pguidType=%s)", this, \_toLog(pguidType));

\*pguidType = GUID\_NULL;

return S\_OK;

}

// IVsOutput2

HRESULT get\_RootRelativeURL(/+[out]+/ BSTR \*pbstrRelativePath)

{

mixin(LogCallMix);

return returnError(E\_NOTIMPL);

}

HRESULT get\_Property(in LPCOLESTR szProperty, /+[out]+/ VARIANT \*pvar)

{

mixin(LogCallMix);

string prop = to\_string(szProperty);

if (icmp(prop, "OUTPUTLOC") == 0)

{

pvar.vt = VT\_BSTR;

pvar.bstrVal = allocBSTR(mTarget);

return S\_OK;

}

return returnError(E\_NOTIMPL);

}

}

class VsOutputGroup : DComObject, IVsOutputGroup

{

this(Config cfg)

{

mConfig = cfg;

}

override HRESULT QueryInterface(in IID\* riid, void\*\* pvObject)

{

if(queryInterface!(IVsOutputGroup) (this, riid, pvObject))

return S\_OK;

return super.QueryInterface(riid, pvObject);

}

// These return identical information regardless of cfg setting:

HRESULT get\_CanonicalName(/+[out]+/ BSTR \*pbstrCanonicalName)

{

mixin(LogCallMix);

\*pbstrCanonicalName = allocBSTR(to\_string(VS\_OUTPUTGROUP\_CNAME\_Built));

return S\_OK;

}

HRESULT get\_DisplayName(/+[out]+/ BSTR \*pbstrDisplayName)

{

mixin(LogCallMix);

\*pbstrDisplayName = allocBSTR("Project build target");

return S\_OK;

}

// The results of these will vary based on the configuration:

HRESULT get\_KeyOutput(/+[out]+/ BSTR \*pbstrCanonicalName)

{

mixin(LogCallMix);

string target = makeFilenameAbsolute(mConfig.GetTargetPath(), mConfig.GetProjectDir());

\*pbstrCanonicalName = allocBSTR(target);

return S\_OK;

}

// Back pointer to project cfg:

HRESULT get\_ProjectCfg(/+[out]+/ IVsProjectCfg2 \*ppIVsProjectCfg2)

{

mixin(LogCallMix);

return returnError(E\_NOTIMPL);

}

// The list of outputs. There might be none! Not all files go out

// on every configuration, and a groups files might all be configuration

// dependent!

HRESULT get\_Outputs(in ULONG celt,

/+[in, out, size\_is(celt)]+/ IVsOutput2 \*rgpcfg,

/+[out, optional]+/ ULONG \*pcActual)

{

mixin(LogCallMix);

if(celt >= 1)

{

string target = makeFilenameAbsolute(mConfig.GetTargetPath(), mConfig.GetProjectDir());

\*rgpcfg = addref(newCom!VsOutput(target));

}

if(pcActual)

\*pcActual = 1;

return S\_OK;

}

HRESULT get\_DeployDependencies(in ULONG celt,

/+[in, out, size\_is(celt)]+/ IVsDeployDependency \*rgpdpd,

/+[out, optional]+/ ULONG \*pcActual)

{

mixin(LogCallMix);

return returnError(E\_NOTIMPL);

}

HRESULT get\_Description(/+[out]+/ BSTR \*pbstrDescription)

{

mixin(LogCallMix);

return returnError(E\_NOTIMPL);

}

private:

Config mConfig;

};

///////////////////////////////////////////////////////////////////////

version(hasProfilableConfig)

{

class TargetInfoFactory : DComObject, IClassFactory

{

override HRESULT QueryInterface(in IID\* riid, void\*\* pvObject)

{

if(queryInterface2!(IClassFactory) (this, IID\_IClassFactory, riid, pvObject))

return S\_OK;

return super.QueryInterface(riid, pvObject);

}

override HRESULT CreateInstance(IUnknown UnkOuter, in IID\* riid, void\*\* pvObject)

{

logCall("%s.CreateInstance(riid=%s)", this, \_toLog(riid));

assert(!UnkOuter);

ProfilerTargetInfo pti = newCom!ProfilerTargetInfo(null);

return pti.QueryInterface(riid, pvObject);

}

override HRESULT LockServer(in BOOL fLock)

{

return returnError(E\_NOTIMPL);

}

}

class ProfilerTargetInfo : DComObject, IVsProfilerTargetInfo, IVsProfilerLaunchExeTargetInfo, IMarshal

{

string mPlatform;

string mWorkdir;

string mProgram;

string mArgs;

this(Config cfg)

{

if(cfg)

{

mPlatform = cfg.mPlatform;

mWorkdir = cfg.mProjectOptions.replaceEnvironment(cfg.mProjectOptions.debugworkingdir, cfg);

if(!isAbsolute(mWorkdir))

mWorkdir = cfg.GetProjectDir() ~ "\\" ~ mWorkdir;

mProgram = cfg.mProjectOptions.replaceEnvironment(cfg.mProjectOptions.debugtarget, cfg);

if(!isAbsolute(mProgram))

mProgram = makeFilenameAbsolute(mProgram, mWorkdir);

mArgs = cfg.mProjectOptions.replaceEnvironment(cfg.mProjectOptions.debugarguments, cfg);

}

}

override HRESULT QueryInterface(in IID\* riid, void\*\* pvObject)

{

if(queryInterface!(IVsProfilerTargetInfo) (this, riid, pvObject))

return S\_OK;

if(queryInterface!(IVsProfilerLaunchTargetInfo) (this, riid, pvObject))

return S\_OK;

if(queryInterface!(IVsProfilerLaunchExeTargetInfo) (this, riid, pvObject))

return S\_OK;

if(queryInterface!(IMarshal) (this, riid, pvObject))

return S\_OK;

return super.QueryInterface(riid, pvObject);

}

// IVsProfilerTargetInfo

HRESULT ProcessArchitecture(VSPROFILERPROCESSARCHTYPE\* arch)

{

mixin(LogCallMix2);

if(mPlatform == "x64")

\*arch = ARCH\_X64;

else

\*arch = ARCH\_X86;

return S\_OK;

}

// IVsProfilerLaunchTargetInfo

HRESULT References(SAFEARRAY\* rgbstr)

{

mixin(LogCallMix2);

return S\_OK;

}

HRESULT EnvironmentSettings(SAFEARRAY\* pbstr)

{

mixin(LogCallMix2);

return S\_OK;

}

HRESULT LaunchProfilerFlags(VSPROFILERLAUNCHOPTS\* opts)

{

mixin(LogCallMix2);

\*opts = VSPLO\_NOPROFILE; // to just launch the exe!?

return S\_OK;

}

// IVsProfilerLaunchExeTargetInfo

HRESULT ExecutableArguments(BSTR\* pbstr)

{

mixin(LogCallMix2);

\*pbstr = allocBSTR(mArgs);

return S\_OK;

}

HRESULT ExecutablePath (BSTR\* pbstr)

{

mixin(LogCallMix2);

\*pbstr = allocBSTR(mProgram);

return S\_OK;

}

HRESULT WorkingDirectory (BSTR\* pbstr)

{

mixin(LogCallMix2);

\*pbstr = allocBSTR(mWorkdir[0..$-1]);

return S\_OK;

}

// IMarshall

override HRESULT GetUnmarshalClass(

/+[in]+/ in IID\* riid,

/+[in, unique]+/ in void \*pv,

/+[in]+/ in DWORD dwDestContext,

/+[in, unique]+/ in void \*pvDestContext,

/+[in]+/ in DWORD mshlflags,

/+[out]+/ CLSID \*pCid)

{

mixin(LogCallMixNoRet);

\*cast(GUID\*)pCid = g\_unmarshalTargetInfoCLSID;

return S\_OK;

//return returnError(E\_NOTIMPL);

}

override HRESULT GetMarshalSizeMax(

/+[in]+/ in IID\* riid,

/+[in, unique]+/ in void \*pv,

/+[in]+/ in DWORD dwDestContext,

/+[in, unique]+/ in void \*pvDestContext,

/+[in]+/ in DWORD mshlflags,

/+[out]+/ DWORD \*pSize)

{

mixin(LogCallMixNoRet);

DWORD size = iid.sizeof;

size += int.sizeof + mPlatform.length;

size += int.sizeof + mWorkdir.length;

size += int.sizeof + mProgram.length;

size += int.sizeof + mArgs.length;

\*pSize = size;

return S\_OK;

}

override HRESULT MarshalInterface(

/+[in, unique]+/ IStream pStm,

/+[in]+/ in IID\* riid,

/+[in, unique]+/ in void \*pv,

/+[in]+/ in DWORD dwDestContext,

/+[in, unique]+/ in void \*pvDestContext,

/+[in]+/ in DWORD mshlflags)

{

mixin(LogCallMixNoRet);

HRESULT hr = pStm.Write(cast(void\*)&iid, iid.sizeof, null);

void writeString(string s)

{

int length = s.length;

if(hr == S\_OK)

hr = pStm.Write(&length, length.sizeof, null);

if(hr == S\_OK && length > 0)

hr = pStm.Write(cast(void\*)s.ptr, length, null);

}

writeString(mPlatform);

writeString(mWorkdir);

writeString(mProgram);

writeString(mArgs);

return hr;

}

override HRESULT UnmarshalInterface(

/+[in, unique]+/ IStream pStm,

/+[in]+/ in IID\* riid,

/+[out]+/ void \*\*ppv)

{

mixin(LogCallMix);

GUID miid;

HRESULT hr = pStm.Read(&miid, iid.sizeof, null);

if (hr == S\_OK)

assert(miid == iid);

void readString(ref string str)

{

int length;

if(hr == S\_OK)

hr = pStm.Read(&length, length.sizeof, null);

if(hr == S\_OK)

{

char[] s = new char[length];

hr = pStm.Read(s.ptr, length, null);

if(hr == S\_OK)

str = assumeUnique(s);

}

}

ProfilerTargetInfo pti = newCom!ProfilerTargetInfo(null);

readString(pti.mPlatform);

readString(pti.mWorkdir);

readString(pti.mProgram);

readString(pti.mArgs);

if(hr != S\_OK)

return returnError(hr);

return pti.QueryInterface(riid, ppv);

}

override HRESULT ReleaseMarshalData(/+[in, unique]+/ IStream pStm)

{

mixin(LogCallMix2);

return returnError(E\_NOTIMPL);

}

override HRESULT DisconnectObject(/+[in]+/ in DWORD dwReserved)

{

logCall("%s.DisconnectObject(dwReserved=%s)", this, \_toLog(dwReserved));

return returnError(E\_NOTIMPL);

}

int mExternalReferences;

}

class EnumVsProfilerTargetInfos : DComObject, IEnumVsProfilerTargetInfos

{

Config mConfig;

int mPos;

this(Config cfg)

{

mConfig = cfg;

mPos = 0;

}

override HRESULT QueryInterface(in IID\* riid, void\*\* pvObject)

{

if(queryInterface2!(IEnumVsProfilerTargetInfos) (this, uuid\_IEnumVsProfilerTargetInfos, riid, pvObject))

return S\_OK;

return super.QueryInterface(riid, pvObject);

}

HRESULT Next(in ULONG celt, IVsProfilerTargetInfo \*rgelt, ULONG \*pceltFetched)

{

ULONG fetched = 0;

if(mPos == 0 && celt > 0)

{

\*rgelt = addref(newCom!ProfilerTargetInfo(mConfig));

fetched = 1;

mPos++;

}

if(pceltFetched)

\*pceltFetched = fetched;

return fetched > 0 ? S\_OK : S\_FALSE;

}

HRESULT Skip(in ULONG celt)

{

mPos += celt;

return S\_OK;

}

HRESULT Reset()

{

mPos = 0;

return S\_OK;

}

HRESULT Clone(IEnumVsProfilerTargetInfos \*ppenum)

{

\*ppenum = addref(newCom!EnumVsProfilerTargetInfos(mConfig));

return S\_OK;

}

}

} // version(hasProfilableConfig)

Config GetActiveConfig(IVsHierarchy pHierarchy)

{

if(!pHierarchy)

return null;

auto solutionBuildManager = queryService!(IVsSolutionBuildManager)();

scope(exit) release(solutionBuildManager);

IVsProjectCfg activeCfg;

if(solutionBuildManager.FindActiveProjectCfg(null, null, pHierarchy, &activeCfg) == S\_OK)

{

scope(exit) release(activeCfg);

if(Config cfg = qi\_cast!Config(activeCfg))

return cfg;

}

return null;

}

// This file is part of Visual D

//

// Visual D integrates the D programming language into Visual Studio

// Copyright (c) 2010-2011 by Rainer Schuetze, All Rights Reserved

//

// Distributed under the Boost Software License, Version 1.0.

// See accompanying file LICENSE\_1\_0.txt or copy at <http://www.boost.org/LICENSE_1_0.txt>

module visuald.cppwizard;

import visuald.windows;

import visuald.winctrl;

import visuald.comutil;

import visuald.register;

import visuald.pkgutil;

import visuald.hierutil;

import visuald.logutil;

import visuald.fileutil;

import visuald.stringutil;

import visuald.wmmsg;

import visuald.dpackage;

import visuald.dimagelist;

import c2d.tokutil;

import c2d.cpp2d;

import sdk.win32.commctrl;

import sdk.vsi.vsshell;

import sdk.vsi.vsshell80;

import dte80a = sdk.vsi.dte80a;

import dte80 = sdk.vsi.dte80;

import stdext.file;

import stdext.string;

import std.algorithm;

import std.conv;

import std.string;

import core.thread;

private IVsWindowFrame sWindowFrame;

private        CppWizardPane sWizardPane;

const int kPaneMargin = 0;

const int kBackMargin = 4;

class Cpp2D : Cpp2DConverter

{

override void writemsg(string s)

{

writeToBuildOutputPane(s ~ "\n");

}

}

bool createCppWizardWindow()

{

if(!sWindowFrame)

{

auto pIVsUIShell = ComPtr!(IVsUIShell)(queryService!(IVsUIShell), false);

if(!pIVsUIShell)

return false;

sWizardPane = newCom!CppWizardPane();

const(wchar)\* caption = "Visual D C++ Conversion Wizard"w.ptr;

HRESULT hr;

hr = pIVsUIShell.CreateToolWindow(CTW\_fInitNew, 0, sWizardPane,

&GUID\_NULL, &g\_CppWizardWinCLSID, &GUID\_NULL,

null, caption, null, &sWindowFrame);

if(!SUCCEEDED(hr))

{

sWizardPane = null;

return false;

}

}

return true;

}

bool showCppWizardWindow()

{

if(!createCppWizardWindow())

return false;

if(FAILED(sWindowFrame.Show()))

return false;

BOOL fHandled;

sWizardPane.\_OnSetFocus(0, 0, 0, fHandled);

return fHandled != 0;

}

bool closeCppWizardWindow()

{

sWindowFrame = release(sWindowFrame);

sWizardPane = null;

return true;

}

bool convertSelection(IVsTextView view)

{

IVsTextLines buffer;

if(FAILED(view.GetBuffer(&buffer)) || !buffer)

return false;

scope(exit) release(buffer);

if(!createCppWizardWindow())

return false;

return sWizardPane.runTextConversion(view, buffer, true);

}

class CppWizardWindowBack : Dialog

{

this(Window parent, CppWizardPane pane)

{

mPane = pane;

super(parent);

}

override int WindowProc(HWND hWnd, uint uMsg, WPARAM wParam, LPARAM lParam)

{

BOOL fHandled;

LRESULT rc = mPane.\_WindowProc(hWnd, uMsg, wParam, lParam, fHandled);

if(fHandled)

return rc;

return super.WindowProc(hWnd, uMsg, wParam, lParam);

}

CppWizardPane mPane;

}

class CppWizardPane : DisposingComObject, IVsWindowPane

{

IServiceProvider mSite;

this()

{

\_ReadStateFromRegistry();

}

override HRESULT QueryInterface(in IID\* riid, void\*\* pvObject)

{

if(queryInterface!(IVsWindowPane) (this, riid, pvObject))

return S\_OK;

// avoid debug output

if(\*riid == IVsCodeWindow.iid || \*riid == IServiceProvider.iid || \*riid == IVsTextView.iid)

return E\_NOINTERFACE;

return super.QueryInterface(riid, pvObject);

}

override void Dispose()

{

mSite = release(mSite);

}

HRESULT SetSite(/+[in]+/ IServiceProvider psp)

{

mixin(LogCallMix2);

mSite = release(mSite);

mSite = addref(psp);

return S\_OK;

}

HRESULT CreatePaneWindow(in HWND hwndParent, in int x, in int y, in int cx, in int cy,

/+[out]+/ HWND \*hwnd)

{

mixin(LogCallMix2);

\_wndParent = new Window(hwndParent);

\_wndBack = new CppWizardWindowBack(\_wndParent, this);

BOOL fHandled;

\_OnInitDialog(WM\_INITDIALOG, 0, 0, fHandled);

\_CheckSize();

\_wndBack.setVisible(true);

return S\_OK;

}

HRESULT GetDefaultSize(/+[out]+/ SIZE \*psize)

{

mixin(LogCallMix2);

psize.cx = 300;

psize.cy = 200;

return S\_OK;

}

HRESULT ClosePane()

{

mixin(LogCallMix2);

if(\_wndParent)

{

\_WriteStateToRegistry();

\_wndParent.Dispose();

\_wndParent = null;

\_wndBack = null;

\_wndToolbar = null;

\_wndFilesLabel = null;

\_wndFilesText = null;

\_wndCodeHdrLabel = null;

\_wndCodeHdrText = null;

\_wndReplaceLabel = null;

\_wndReplacePreText = null;

\_wndReplacePostText = null;

\_wndInputTypeLabel = null;

\_wndInputType = null;

\_wndKeywordPrefixLabel = null;

\_wndKeywordPrefixText = null;

\_wndPackagePrefixLabel = null;

\_wndPackagePrefixText = null;

\_wndOutputDirLabel = null;

\_wndOutputDirText = null;

\_wndInputDirLabel = null;

\_wndInputDirText = null;

\_wndVersionsLabel = null;

\_wndVersionsText = null;

\_wndExpansionsLabel = null;

\_wndExpansionsText = null;

\_wndValueTypesLabel = null;

\_wndValueTypesText = null;

\_wndClassTypesLabel = null;

\_wndClassTypesText = null;

\_wndWriteIntermediate = null;

\_wndLoad = null;

\_wndSave = null;

\_wndConvert = null;

mDlgFont = deleteDialogFont(mDlgFont);

if(\_himlToolbar)

ImageList\_Destroy(\_himlToolbar);

}

return S\_OK;

}

HRESULT LoadViewState(/+[in]+/ IStream pstream)

{

mixin(LogCallMix2);

return returnError(E\_NOTIMPL);

}

HRESULT SaveViewState(/+[in]+/ IStream pstream)

{

mixin(LogCallMix2);

return returnError(E\_NOTIMPL);

}

HRESULT TranslateAccelerator(MSG\* msg)

{

if(msg.message == WM\_TIMER)

\_CheckSize();

if(msg.message == WM\_TIMER || msg.message == WM\_SYSTIMER)

return E\_NOTIMPL; // do not flood debug output

logMessage("TranslateAccelerator", msg.hwnd, msg.message, msg.wParam, msg.lParam);

BOOL fHandled;

HRESULT hrRet = \_HandleMessage(msg.hwnd, msg.message, msg.wParam, msg.lParam, fHandled);

if(fHandled)

return hrRet;

return E\_NOTIMPL;

}

///////////////////////////////////////////////////////////////////

// the following has been ported from the FlatSolutionExplorer project

private:

C2DIni \_options;

Window \_wndParent;

CppWizardWindowBack \_wndBack;

ToolBar \_wndToolbar;

HIMAGELIST \_himlToolbar;

HFONT mDlgFont;

Label \_wndInputTypeLabel;

ComboBox \_wndInputType;

Label \_wndKeywordPrefixLabel;

Text \_wndKeywordPrefixText;

Label \_wndPackagePrefixLabel;

Text \_wndPackagePrefixText;

Label \_wndOutputDirLabel;

Text \_wndOutputDirText;

Label \_wndInputDirLabel;

Text \_wndInputDirText;

Label \_wndFilesLabel;

MultiLineText \_wndFilesText;

Label \_wndCodeHdrLabel;

MultiLineText \_wndCodeHdrText;

Label \_wndReplaceLabel;

MultiLineText \_wndReplacePreText;

MultiLineText \_wndReplacePostText;

Label \_wndVersionsLabel;

MultiLineText \_wndVersionsText;

Label \_wndExpansionsLabel;

MultiLineText \_wndExpansionsText;

Label \_wndValueTypesLabel;

MultiLineText \_wndValueTypesText;

Label \_wndClassTypesLabel;

MultiLineText \_wndClassTypesText;

CheckBox \_wndWriteIntermediate;

Button \_wndLoad;

Button \_wndSave;

Button \_wndConvert;

static HINSTANCE getInstance() { return Widget.getInstance(); }

int \_WindowProc(HWND hWnd, uint uMsg, WPARAM wParam, LPARAM lParam, ref BOOL fHandled)

{

if(uMsg != WM\_NOTIFY)

logMessage("\_WindowProc", hWnd, uMsg, wParam, lParam);

return \_HandleMessage(hWnd, uMsg, wParam, lParam, fHandled);

}

int \_HandleMessage(HWND hWnd, uint uMsg, WPARAM wParam, LPARAM lParam, ref BOOL fHandled)

{

switch(uMsg)

{

case WM\_CREATE:

case WM\_INITDIALOG:

return \_OnInitDialog(uMsg, wParam, lParam, fHandled);

case WM\_DESTROY:

return \_OnDestroy(uMsg, wParam, lParam, fHandled);

case WM\_SIZE:

if(hWnd == \_wndBack.hwnd)

return \_OnSize(uMsg, wParam, lParam, fHandled);

break;

case WM\_KEYDOWN:

case WM\_SYSKEYDOWN:

return \_OnKeyDown(uMsg, wParam, lParam, fHandled);

case WM\_NCACTIVATE:

case WM\_SETFOCUS:

return \_OnSetFocus(uMsg, wParam, lParam, fHandled);

case WM\_COMMAND:

ushort id = LOWORD(wParam);

ushort code = HIWORD(wParam);

if(id == IDC\_WIZ\_INPUTTPYE && code == CBN\_SELCHANGE)

\_UpdateEnableState();

if(code == BN\_CLICKED)

{

switch(id)

{

case IDC\_WIZ\_LOAD:

string file = getOpenFileDialog(hWnd, "Load Conversion Config", "", "Conversion Files|\*.c2d|");

if(file.length)

{

tryWithExceptionToBuildOutputPane( (){

\_options.readFromFile(file);

}, file);

\_OptionsToDialog();

}

return 0;

case IDC\_WIZ\_SAVE:

string file = getSaveFileDialog(hWnd, "Save Conversion Config", "", "Conversion Files|\*.c2d|");

if(file.length)

{

\_DialogToOptions();

tryWithExceptionToBuildOutputPane( (){

\_options.writeToFile(file);

}, file);

}

return 0;

case IDC\_WIZ\_CONVERT:

runConversion();

//sWindowFrame.Hide();

return 0;

default:

break;

}

}

break;

/+

case WM\_NCCALCSIZE:

return \_OnCalcSize(uMsg, wParam, lParam, fHandled);

case WM\_CONTEXTMENU:

return \_OnContextMenu(uMsg, wParam, lParam, fHandled);

case WM\_NOTIFY:

if (nmhdr.idFrom == IDC\_TOOLBAR && nmhdr.code == TBN\_GETINFOTIP)

return \_OnToolbarGetInfoTip(wParam, nmhdr, fHandled);

break;

+/

default:

break;

}

return 0;

}

LRESULT \_OnInitDialog(UINT uiMsg, WPARAM wParam, LPARAM lParam, ref BOOL fHandled)

{

if(\_wndInputTypeLabel)

return S\_OK;

updateEnvironmentFont();

if(!mDlgFont)

mDlgFont = newDialogFont();

\_wndInputTypeLabel = new Label(\_wndBack, "&Convert:", -1);

\_wndInputType = new ComboBox(\_wndBack, [ "Input files", "Current Document",

"Current Selection" ], false, IDC\_WIZ\_INPUTTPYE);

\_wndFilesLabel = new Label(\_wndBack, "Fi&les and directories:", -1);

\_wndFilesText = new MultiLineText(\_wndBack, "", IDC\_WIZ\_INPUTFILES);

\_wndCodeHdrLabel = new Label(\_wndBack, "Source Code Header:", -1);

\_wndCodeHdrText = new MultiLineText(\_wndBack, "", IDC\_WIZ\_CODEHDR);

\_wndKeywordPrefixLabel = new Label(\_wndBack, "Keyword Prefix:", -1);

\_wndKeywordPrefixText = new Text(\_wndBack, "", IDC\_WIZ\_KEYWORDPREFIX);

\_wndPackagePrefixLabel = new Label(\_wndBack, "Package Prefix:", -1);

\_wndPackagePrefixText = new Text(\_wndBack, "", IDC\_WIZ\_PACKAGEPREFIX);

\_wndOutputDirLabel = new Label(\_wndBack, "Output Dir:", -1);

\_wndOutputDirText = new Text(\_wndBack, "", IDC\_WIZ\_OUTPUTDIR);

\_wndInputDirLabel = new Label(\_wndBack, "Input Dir:", -1);

\_wndInputDirText = new Text(\_wndBack, "", IDC\_WIZ\_INPUTDIR);

\_wndReplaceLabel = new Label(\_wndBack, "Pre and Post Token Re&placements (pattern => replacement):", -1);

\_wndReplacePreText = new MultiLineText(\_wndBack, "", IDC\_WIZ\_REPLACEPRE);

\_wndReplacePostText = new MultiLineText(\_wndBack, "", IDC\_WIZ\_REPLACEPOST);

\_wndWriteIntermediate = new CheckBox(\_wndBack, "Write intermediate files", IDC\_WIZ\_WRITEINTERMED);

\_wndVersionsLabel = new Label(\_wndBack, "Version Conditionals:", -1);

\_wndVersionsText = new MultiLineText(\_wndBack, "", IDC\_WIZ\_VERSIONS);

\_wndExpansionsLabel = new Label(\_wndBack, "Preprocessor expansions:", -1);

\_wndExpansionsText = new MultiLineText(\_wndBack, "", IDC\_WIZ\_EXPANSIONS);

\_wndValueTypesLabel = new Label(\_wndBack, "Value types:", -1);

\_wndValueTypesText = new MultiLineText(\_wndBack, "", IDC\_WIZ\_VALUETYPES);

\_wndClassTypesLabel = new Label(\_wndBack, "Reference types:", -1);

\_wndClassTypesText = new MultiLineText(\_wndBack, "", IDC\_WIZ\_CLASSTYPES);

\_wndLoad = new Button(\_wndBack, "&Load", IDC\_WIZ\_LOAD);

\_wndSave = new Button(\_wndBack, "&Save", IDC\_WIZ\_SAVE);

\_wndConvert = new Button(\_wndBack, "&Convert", IDC\_WIZ\_CONVERT);

\_OptionsToDialog();

RECT r;

\_wndBack.GetClientRect(&r);

\_layoutViews(r.right - r.left, r.bottom - r.top);

// \_InitializeToolbar();

return S\_OK;

}

LRESULT \_OnDestroy(UINT uiMsg, WPARAM wParam, LPARAM lParam, ref BOOL fHandled)

{

if (\_himlToolbar)

{

\_wndToolbar.SendMessage(TB\_SETIMAGELIST, 0, cast(LPARAM)null);

ImageList\_Destroy(\_himlToolbar);

\_himlToolbar = null;

}

fHandled = TRUE;

// return CComCompositeControl<CFlatSolutionExplorer>::OnDestroy(uiMsg, wParam, lParam, fHandled);

return 0;

}

LRESULT \_OnKeyDown(UINT uiMsg, WPARAM wParam, LPARAM lParam, ref BOOL fHandled)

{

UINT vKey = LOWORD(wParam);

switch(vKey)

{

case VK\_ESCAPE:

sWindowFrame.Hide();

break;

default:

break;

}

return 0;

}

void \_CheckSize()

{

RECT r, br;

\_wndParent.GetClientRect(&r);

\_wndBack.GetClientRect(&br);

if(br.right - br.left != r.right - r.left - 2\*kPaneMargin ||

br.bottom - br.top != r.bottom - r.top - 2\*kPaneMargin)

\_wndBack.setRect(kPaneMargin, kPaneMargin,

r.right - r.left - 2\*kPaneMargin, r.bottom - r.top - 2\*kPaneMargin);

}

// Adjust child control sizes

void \_layoutViews(int cw, int ch)

{

int top = kBackMargin; // kToolBarAtTop ? kToolBarHeight : 1;

int bot = ch - kBackMargin;

int lineh = 16;

int combh = 20;

int lblspacing = 1;

int spacing = 3;

int btnw = 80;

int btnh = 24;

int x = kBackMargin;

int w = cw - 2 \* kBackMargin;

// ##InputType######## ##KWPrefex###

// ##OutputDir######## X Add2Startup

// ##InputDir#######################

// +-Files-------------------------\

// \-------------------------------+

// +-Replace Pre--\+-Replace Post--\

// \--------------+\---------------+

// +-Versions-----\+-Expansions----\

// \----------.---+\---------------+

// +-ValueTypes---\+-RefTypes------\

// \----------.---+\---------------+

// Load Save Conv

int lblwidth = 64;

int kwpwidth = 150;

\_wndInputTypeLabel.setRect(x, top + 2, lblwidth, lineh);

\_wndInputType.setRect(x + lblwidth, top, w - lblwidth - kwpwidth - 10, combh);

top += combh + 2 + spacing;

\_wndInputDirLabel.setRect(x, top, lblwidth, lineh);

\_wndInputDirText.setRect(x + lblwidth, top, w - lblwidth, lineh);

top += lineh + spacing;

\_wndConvert.setRect(x + w - btnw - 0 \* (spacing + btnw), bot - btnh, btnw, btnh);

\_wndSave .setRect(x + w - btnw - 1 \* (spacing + btnw), bot - btnh, btnw, btnh);

\_wndLoad .setRect(x + w - btnw - 2 \* (spacing + btnw), bot - btnh, btnw, btnh);

bot -= btnh + spacing + spacing;

int plblwidth = 82;

int tw = max(0, w - spacing) / 2;

\_wndKeywordPrefixLabel.setRect(x, bot - lineh, plblwidth, lineh);

\_wndKeywordPrefixText.setRect(x + plblwidth, bot - lineh, tw - plblwidth - spacing, lineh);

\_wndPackagePrefixLabel.setRect(x + tw, bot - lineh, plblwidth, lineh);

\_wndPackagePrefixText.setRect(x + tw + plblwidth, bot - lineh, tw - plblwidth - spacing, lineh);

bot -= lineh + spacing;

//                \_wndLookIn .setRect(x, bot - combh, w, combh); bot -= combh + lblspacing;

//                \_wndLookInLabel .setRect(x, bot - lineh, w, lineh); bot -= lineh + spacing;

int th = max(0, bot - top - (4 \* spacing + lineh)) / 4;

int txth = max(0, th - lineh + lblspacing - spacing);

\_wndCodeHdrLabel.setRect(x + tw, top, w - tw, lineh);

\_wndFilesLabel.setRect(x, top, w - spacing, lineh); top += lineh + lblspacing;

\_wndCodeHdrText.setRect(x + tw, top, w - tw, txth);

\_wndFilesText.setRect(x, top, tw - spacing, txth); top += txth + spacing;

\_wndOutputDirLabel.setRect(x, top, lblwidth, lineh);

\_wndOutputDirText.setRect(x + lblwidth, top, w - kwpwidth - lblwidth - 10, lineh);

\_wndWriteIntermediate.setRect(w - kwpwidth, top, kwpwidth, lineh);

top += lineh + spacing + spacing;

\_wndVersionsLabel.setRect(x + w - tw, top, tw, lineh);

\_wndExpansionsLabel.setRect(x, top, tw, lineh);

top += lineh + lblspacing;

\_wndVersionsText.setRect(x + w - tw, top, tw, txth);

\_wndExpansionsText.setRect(x, top, tw, txth);

top += txth + spacing;

\_wndReplaceLabel.setRect(x, top, w, lineh); top += lineh + lblspacing;

\_wndReplacePreText.setRect(x, top, tw, txth);

\_wndReplacePostText.setRect(x + w - tw, top, tw, txth); top += txth + spacing;

\_wndValueTypesLabel.setRect(x, top, tw, lineh);

\_wndClassTypesLabel.setRect(x + w - tw, top, tw, lineh);

top += lineh + lblspacing;

\_wndValueTypesText.setRect(x, top, tw, bot - top);

\_wndClassTypesText.setRect(x + w - tw, top, tw, bot - top);

}

LRESULT \_OnSize(UINT uiMsg, WPARAM wParam, LPARAM lParam, ref BOOL fHandled)

{

int cx = LOWORD(lParam);

int cy = HIWORD(lParam);

\_layoutViews(cx, cy);

return 0;

}

LRESULT \_OnSetFocus(UINT uiMsg, WPARAM wParam, LPARAM lParam, ref BOOL fHandled)

{

// Skip the CComCompositeControl handling

// CComControl<CFlatSolutionExplorer, CAxDialogImpl<CFlatSolutionExplorer>>::OnSetFocus(uiMsg, wParam, lParam, fHandled);

if(\_wndFilesText)

{

//\_wndFilesText.SetFocus();

//\_wndFilesText.SendMessage(EM\_SETSEL, 0, cast(LPARAM)-1);

fHandled = TRUE;

}

return 0;

}

void \_UpdateEnableState()

{

int sel = \_wndInputType.getSelection();

bool files = (sel == 0);

\_wndWriteIntermediate.setEnabled(files);

\_wndOutputDirText.setEnabled(files);

\_wndInputDirText.setEnabled(files);

\_wndFilesText.setEnabled(files);

\_wndCodeHdrText.setEnabled(files);

}

void \_OptionsToDialog()

{

\_wndWriteIntermediate.setChecked(\_options.writeIntermediate);

\_wndInputType.setSelection(\_options.inputType);

\_wndFilesText.setText(\_options.inputFiles);

\_wndCodeHdrText.setText(\_options.codePrefix);

\_wndReplacePreText.setText(\_options.replaceTokenPre);

\_wndReplacePostText.setText(\_options.replaceTokenPost);

\_wndKeywordPrefixText.setText(\_options.keywordPrefix);

\_wndPackagePrefixText.setText(\_options.packagePrefix);

\_wndVersionsText.setText(\_options.versionDefines);

\_wndExpansionsText.setText(\_options.expandConditionals);

\_wndValueTypesText.setText(\_options.userValueTypes);

\_wndClassTypesText.setText(\_options.userClassTypes);

\_wndOutputDirText.setText(\_options.outputDir);

\_wndInputDirText.setText(\_options.inputDir);

\_UpdateEnableState();

}

void \_DialogToOptions()

{

\_options.inputType = \_wndInputType.getSelection();

\_options.writeIntermediate = \_wndWriteIntermediate.isChecked();

\_options.inputFiles = \_wndFilesText.getText();

\_options.codePrefix = \_wndCodeHdrText.getText();

\_options.replaceTokenPre = \_wndReplacePreText.getText();

\_options.replaceTokenPost = \_wndReplacePostText.getText();

\_options.keywordPrefix = \_wndKeywordPrefixText.getText();

\_options.packagePrefix = \_wndPackagePrefixText.getText();

\_options.versionDefines = \_wndVersionsText.getText();

\_options.expandConditionals = \_wndExpansionsText.getText();

\_options.userValueTypes = \_wndValueTypesText.getText();

\_options.userClassTypes = \_wndClassTypesText.getText();

\_options.outputDir = \_wndOutputDirText.getText();

\_options.inputDir = \_wndInputDirText.getText();

}

RegKey \_GetCurrentRegKey(bool write)

{

GlobalOptions opt = Package.GetGlobalOptions();

opt.getRegistryRoot();

wstring regPath = opt.regUserRoot ~ regPathToolsOptions;

regPath ~= "\\WizardWindow"w;

return new RegKey(opt.hUserKey, regPath, write);

}

bool \_WriteStateToRegistry()

{

try

{

\_DialogToOptions();

scope RegKey keyWinOpts = \_GetCurrentRegKey(true);

string s = \_options.writeToText();

keyWinOpts.Set("Options"w, to!wstring(s));

}

catch(Exception e)

{

return false;

}

return true;

}

bool \_ReadStateFromRegistry()

{

try

{

scope RegKey keyWinOpts = \_GetCurrentRegKey(false);

wstring s = keyWinOpts.GetString("Options"w, ""w);

\_options = \_options.init;

\_options.readFromText(to!string(s));

}

catch(Exception e)

{

return false;

}

return true;

}

string mLastCText;

string mLastDText;

bool runTextConversion()

{

IVsTextView view;

scope(exit) release(view);

IVsTextLines buffer = GetCurrentTextBuffer(&view);

if(!buffer)

return false;

scope(exit) release(buffer);

if(!view)

return false;

return runTextConversion(view, buffer, \_options.inputType != 1);

}

bool runTextConversion(IVsTextView view, IVsTextLines buffer, bool selection)

{

int startLine, startCol;

int endLine, endCol;

HRESULT hr;

if(selection)

hr = GetSelectionForward(view, &startLine, &startCol, &endLine, &endCol);

else

hr = buffer.GetLastLineIndex(&endLine, &endCol);

if(hr != S\_OK)

return false;

BSTR text;

if(buffer.GetLineText(startLine, startCol, endLine, endCol, &text) != S\_OK)

return false;

string txt = detachBSTR(text);

if(txt == mLastDText)

txt = mLastCText;

auto c2d = new Cpp2D;

\_options.toC2DOptions(/\*c2d.cpp2d.\*/options);

string ntxt = c2d.main(txt);

if(ntxt !is null && txt != ntxt)

{

mLastCText = txt;

mLastDText = ntxt;

wstring wntxt = to!wstring(ntxt);

TextSpan changedSpan;

if(buffer.ReplaceLines(startLine, startCol, endLine, endCol,

wntxt.ptr, wntxt.length, &changedSpan) != S\_OK)

return false;

}

return true;

}

bool runFileConversion()

{

void run()

{

\_options.toC2DOptions(/\*c2d.cpp2d.\*/options);

string[] filespecs = tokenizeArgs(\_options.inputFiles);

string[] files = expandFileList(filespecs, \_options.inputDir);

try

{

auto c2d = new Cpp2D;

c2d.main(files);

}

catch(Throwable e)

{

string msg = e.toString();

writeToBuildOutputPane(msg);

}

}

clearOutputPane();

auto thrd = new Thread(&run);

thrd.start();

return true;

}

bool runConversion()

{

try

{

\_DialogToOptions();

if(\_options.inputType == 0)

return runFileConversion();

return runTextConversion();

}

catch(Throwable e)

{

string msg = e.toString();

writeToBuildOutputPane(msg);

logCall("EXCEPTION: " ~ msg);

return false;

}

}

}

// This file is part of Visual D

//

// Visual D integrates the D programming language into Visual Studio

// Copyright (c) 2010 by Rainer Schuetze, All Rights Reserved

//

// Distributed under the Boost Software License, Version 1.0.

// See accompanying file LICENSE\_1\_0.txt or copy at <http://www.boost.org/LICENSE_1_0.txt>

module visuald.dimagelist;

import stdext.string;

mixin(extractDefines(import("resources.h")));

const kImageDSource = 0;

const kImageProject = 1;

const kImageFolderClosed = 2;

const kImageFolderOpened = 3;

const kImageResource = 4;

const kImageDocument = 5;

const kImageScript = 6;

const kImageDisabled = 7;

const IDC\_TOOLBAR = 1010;

const IDC\_FILEWHEEL = 1011;

const IDC\_FILELIST = 1012;

const IDC\_FILELISTHDR = 1013;

const IDC\_FANINLIST = 1014;

const IDC\_FANOUTLIST = 1015;

const IDC\_FINDTEXT = 2000;

const IDC\_REPLACETEXT = 2001;

const IDC\_FINDMATCHCASE = 2002;

const IDC\_REPLACECASE = 2003;

const IDC\_FINDLOOKIN = 2004;

const IDC\_FINDDIRECTION = 2005;

const IDC\_FINDNEXT = 2006;

const IDC\_REPLACE = 2007;

const IDC\_REPLACEALL = 2008;

const IDC\_FINDCLOSE = 2009;

const IDC\_FINDMATCHBRACES = 2010;

const IDC\_FINDINCCOMMENT = 2011;

const IDC\_WIZ\_WRITEINTERMED = 2020;

const IDC\_WIZ\_INPUTTPYE = 2021;

const IDC\_WIZ\_INPUTFILES = 2022;

const IDC\_WIZ\_KEYWORDPREFIX = 2023;

const IDC\_WIZ\_REPLACEPRE = 2024;

const IDC\_WIZ\_REPLACEPOST = 2025;

const IDC\_WIZ\_VERSIONS = 2026;

const IDC\_WIZ\_EXPANSIONS = 2027;

const IDC\_WIZ\_VALUETYPES = 2028;

const IDC\_WIZ\_CLASSTYPES = 2029;

const IDC\_WIZ\_OUTPUTDIR = 2030;

const IDC\_WIZ\_LOAD = 2031;

const IDC\_WIZ\_SAVE = 2032;

const IDC\_WIZ\_CONVERT = 2033;

const IDC\_WIZ\_INPUTDIR = 2034;

const IDC\_WIZ\_PACKAGEPREFIX = 2035;

const IDC\_WIZ\_CODEHDR = 2036;

// menu ID

const IDM\_COLUMNLISTBASE = 0x100;

// Miscellaneous IDs

const ID\_SUBCLASS\_HDR = 0x100;

// entries in the image list "completionset.bmp" through the envireonment

enum CSIMG\_PROT\_PUBLIC = 0;

enum CSIMG\_PROT\_LETTER = 1;

enum CSIMG\_PROT\_BRIGHT = 2;

enum CSIMG\_PROT\_PROTECTED = 3;

enum CSIMG\_PROT\_PRIVATE = 4;

enum CSIMG\_PROT\_LINK = 5;

enum CSIMG\_CLASS = 0; // combine with CSIMG\_PROT for modifier

enum CSIMG\_PACKAGE = 6;

enum CSIMG\_DELEGATE = 12;

enum CSIMG\_ENUM = 18;

enum CSIMG\_ENUMMEMBER = 24;

enum CSIMG\_BLITZ = 30;

enum CSIMG\_UNKNOWN1 = 36;

enum CSIMG\_FIELD = 42;

enum CSIMG\_INTERFACE = 48;

enum CSIMG\_UNKNOWN2 = 54;

enum CSIMG\_UNKNOWN3 = 60;

enum CSIMG\_UNKNOWN4 = 66;

enum CSIMG\_MEMBER = 72;

enum CSIMG\_MEMBERS = 78;

enum CSIMG\_UNKNOWN5 = 84;

enum CSIMG\_NAMESPACE = 90;

enum CSIMG\_UNKNOWN6 = 96;

enum CSIMG\_PROPERTY = 102;

enum CSIMG\_STRUCT = 108;

enum CSIMG\_TEMPLATE = 114;

enum CSIMG\_UNKNOWN7 = 120;

enum CSIMG\_UNION = 126;

enum CSIMG\_STRUCT3 = 132;

enum CSIMG\_FIELD2 = 138;

enum CSIMG\_STRUCT4 = 144;

enum CSIMG\_UNKNOWN8 = 150;

enum CSIMG\_JMEMBER = 156;

enum CSIMG\_JFIELD = 162;

enum CSIMG\_JSTRUCT = 168;

enum CSIMG\_JNAMESPACE = 174;

enum CSIMG\_JINTERFACE = 180;

enum CSIMG\_STOP = 186; // series of single bitmaps follow

enum CSIMG\_DMODULE = 194;

enum CSIMG\_DFOLDER = 201;

enum CSIMG\_KEYWORD = 206;

// This file is part of Visual D

//

// Visual D integrates the D programming language into Visual Studio

// Copyright (c) 2010 by Rainer Schuetze, All Rights Reserved

//

// Distributed under the Boost Software License, Version 1.0.

// See accompanying file LICENSE\_1\_0.txt or copy at <http://www.boost.org/LICENSE_1_0.txt>

module visuald.dlangsvc;

// import diamond;

import visuald.comutil;

import visuald.logutil;

import visuald.hierutil;

import visuald.fileutil;

import visuald.stringutil;

import visuald.pkgutil;

import visuald.dpackage;

import visuald.dimagelist;

import visuald.expansionprovider;

import visuald.completion;

import visuald.intellisense;

import visuald.searchsymbol;

import visuald.viewfilter;

import visuald.colorizer;

import visuald.windows;

import visuald.simpleparser;

import visuald.config;

import visuald.vdserverclient;

import visuald.vdextensions;

version = VDServer;

//version = DEBUG\_GC;

//version = TWEAK\_GC;

//import rsgc.gc;

version(TWEAK\_GC) {

import rsgc.gcstats;

import core.memory;

extern (C) GCStats gc\_stats();

}

import vdc.lexer;

import vdc.ivdserver;

static import vdc.util;

import stdext.array;

import stdext.string;

import stdext.path;

import std.string;

import std.ascii;

import std.utf;

import std.conv;

import std.algorithm;

import std.array;

import std.datetime;

import std.exception;

import std.parallelism;

import sdk.port.vsi;

import sdk.vsi.textmgr;

import sdk.vsi.textmgr2;

import sdk.vsi.textmgr90;

import sdk.vsi.vsshell;

import sdk.vsi.vsshell80;

import sdk.vsi.singlefileeditor;

import sdk.vsi.fpstfmt;

import sdk.vsi.stdidcmd;

import sdk.vsi.vsdbgcmd;

import sdk.vsi.vsdebugguids;

import sdk.vsi.msdbg;

version = threadedOutlining;

///////////////////////////////////////////////////////////////////////////////

\_\_gshared Lexer dLex;

///////////////////////////////////////////////////////////////////////////////

class LanguageService : DisposingComObject,

IVsLanguageInfo,

IVsLanguageDebugInfo,

IVsLanguageDebugInfo2,

IVsLanguageDebugInfoRemap,

IVsProvideColorableItems,

IVsLanguageContextProvider,

IServiceProvider,

// ISynchronizeInvoke,

IVsDebuggerEvents,

IVsFormatFilterProvider,

IVsOutliningCapableLanguage,

IVsUpdateSolutionEvents

{

static const GUID iid = g\_languageCLSID;

this(Package pkg)

{

//mPackage = pkg;

mUpdateSolutionEvents = newCom!UpdateSolutionEvents(this);

}

~this()

{

}

@property VDServerClient vdServerClient()

{

if(!mVDServerClient)

{

mVDServerClient = new VDServerClient;

mVDServerClient.start();

}

return mVDServerClient;

}

override HRESULT QueryInterface(in IID\* riid, void\*\* pvObject)

{

if(queryInterface!(IVsLanguageInfo) (this, riid, pvObject))

return S\_OK;

if(queryInterface!(IVsProvideColorableItems) (this, riid, pvObject))

return S\_OK;

if(queryInterface!(IVsLanguageDebugInfo) (this, riid, pvObject))

return S\_OK;

if(queryInterface!(IVsLanguageDebugInfo2) (this, riid, pvObject))

return S\_OK;

if(queryInterface!(IVsLanguageDebugInfoRemap) (this, riid, pvObject))

return S\_OK;

if(queryInterface!(IVsDebuggerEvents) (this, riid, pvObject))

return S\_OK;

// delegated to mUpdateSolutionEvents

//                if(queryInterface!(IVsUpdateSolutionEvents) (this, riid, pvObject))

//                        return S\_OK;

//                if(queryInterface!(IVsFormatFilterProvider) (this, riid, pvObject))

//                        return S\_OK;

if(queryInterface!(IVsOutliningCapableLanguage) (this, riid, pvObject))

return S\_OK;

return super.QueryInterface(riid, pvObject);

}

void stopAllParsing()

{

if(Source.parseTaskPool)

{

//Source.parseTaskPool.finish();

//Source.parseTaskPool.wait();

Source.parseTaskPool.stop();

}

}

// IDisposable

override void Dispose()

{

stopAllParsing();

closeSearchWindow();

setDebugger(null);

foreach(Source src; mSources)

src.Release();

mSources = mSources.init;

foreach(CodeWindowManager mgr; mCodeWinMgrs)

mgr.Release();

mCodeWinMgrs = mCodeWinMgrs.init;

if(mVDServerClient)

mVDServerClient.shutDown();

if(mUpdateSolutionEventsCookie != VSCOOKIE\_NIL)

{

auto solutionBuildManager = queryService!(IVsSolutionBuildManager)();

if(solutionBuildManager)

{

scope(exit) release(solutionBuildManager);

solutionBuildManager.UnadviseUpdateSolutionEvents(mUpdateSolutionEventsCookie);

mUpdateSolutionEventsCookie = VSCOOKIE\_NIL;

}

}

cdwLastSource = null;

mLastActiveView = null;

}

// IVsLanguageInfo //////////////////////////////////////

override HRESULT GetCodeWindowManager(IVsCodeWindow pCodeWin, IVsCodeWindowManager\* ppCodeWinMgr)

{

IVsTextLines buffer;

if(pCodeWin.GetBuffer(&buffer) == S\_OK)

{

Source src = GetSource(buffer);

CodeWindowManager mgr = newCom!CodeWindowManager(this, pCodeWin, src);

mCodeWinMgrs ~= addref(mgr);

\*ppCodeWinMgr = addref(mgr);

}

return S\_OK;

}

override HRESULT GetColorizer(IVsTextLines pBuffer, IVsColorizer\* ppColorizer)

{

if(mUpdateSolutionEventsCookie == VSCOOKIE\_NIL)

{

auto solutionBuildManager = queryService!(IVsSolutionBuildManager)();

if(solutionBuildManager)

{

scope(exit) release(solutionBuildManager);

solutionBuildManager.AdviseUpdateSolutionEvents(mUpdateSolutionEvents, &mUpdateSolutionEventsCookie);

}

}

Source src = GetSource(pBuffer);

\*ppColorizer = addref(src.mColorizer);

return S\_OK;

}

override HRESULT GetFileExtensions(BSTR\* pbstrExtensions)

{

return E\_NOTIMPL;

}

override HRESULT GetLanguageName(BSTR\* bstrName)

{

return E\_NOTIMPL;

}

// IVsLanguageDebugInfo //////////////////////////////////////

override HRESULT GetLanguageID(IVsTextBuffer pBuffer, in int iLine, in int iCol, GUID\* pguidLanguageID)

{

\*pguidLanguageID = g\_languageCLSID;

return S\_OK;

}

// obsolete

override HRESULT GetLocationOfName(in LPCOLESTR pszName, BSTR\* pbstrMkDoc, TextSpan\* pspanLocation)

{

mixin(LogCallMix);

\*pbstrMkDoc = null;

return E\_NOTIMPL;

}

override HRESULT GetNameOfLocation(IVsTextBuffer pBuffer, in int iLine, in int iCol, BSTR\* pbstrName, int\* piLineOffset)

{

mixin(LogCallMix);

/\*

string fname;

if(IPersistFileFormat fileFormat = qi\_cast!IPersistFileFormat(pBuffer))

{

scope(exit) release(fileFormat);

uint format;

LPOLESTR filename;

if(fileFormat.GetCurFile(&filename, &format) == S\_OK)

fname = detachOLESTR(filename);

}

\*pbstrName = allocBSTR(fname);

\*/

\*pbstrName = null;

\*piLineOffset = 0;

return S\_OK;

}

override HRESULT GetProximityExpressions(IVsTextBuffer pBuffer, in int iLine, in int iCol, in int cLines, IVsEnumBSTR\* ppEnum)

{

auto text = ComPtr!(IVsTextLines)(pBuffer);

if(!text)

return E\_FAIL;

Source src = GetSource(text);

if(!src)

return E\_FAIL;

\*ppEnum = addref(newCom!EnumProximityExpressions(src, iLine, iCol, cLines));

return S\_OK;

}

override HRESULT IsMappedLocation(IVsTextBuffer pBuffer, in int iLine, in int iCol)

{

mixin(LogCallMix);

return S\_FALSE;

}

override HRESULT ResolveName(in LPCOLESTR pszName, in uint dwFlags, IVsEnumDebugName\* ppNames)

{

mixin(LogCallMix);

return E\_NOTIMPL;

}

override HRESULT ValidateBreakpointLocation(IVsTextBuffer pBuffer, in int iLine, in int iCol, TextSpan\* pCodeSpan)

{

pCodeSpan.iStartLine = iLine;

pCodeSpan.iStartIndex = 0;

pCodeSpan.iEndLine = iLine;

pCodeSpan.iEndIndex = 0;

return S\_OK;

}

// IVsLanguageDebugInfo2 //////////////////////////////////////

HRESULT QueryCommonLanguageBlock(

/+[in]+/ IVsTextBuffer pBuffer, //code buffer containing a break point

in int iLine, //line for a break point

in int iCol, //column for a break point

in DWORD dwFlag, //common language block being queried. see LANGUAGECOMMONBLOCK

/+[out]+/ BOOL \*pfInBlock) //true if iLine and iCol is inside common language block;otherwise, false;

{

mixin(LogCallMix);

return E\_NOTIMPL;

}

HRESULT ValidateInstructionpointLocation(

/+[in]+/ IVsTextBuffer pBuffer, //code buffer containing an instruction point(IP)

in int iLine, //line for the existing IP

in int iCol, //column for the existing IP

/+[out]+/ TextSpan \*pCodeSpan) //new IP code span

{

mixin(LogCallMix);

return E\_NOTIMPL;

}

HRESULT QueryCatchLineSpan(

/+[in]+/ IVsTextBuffer pBuffer, //code buffer containing a break point

in int iLine, //line for a break point

in int iCol, //column for a break point

/+[out]+/ BOOL \*pfIsInCatch,

/+[out]+/ TextSpan \*ptsCatchLine)

{

mixin(LogCallMix);

return E\_NOTIMPL;

}

// IVsLanguageDebugInfoRemap //////////////////////////////////////

HRESULT RemapBreakpoint(/+[in]+/ IUnknown pUserBreakpointRequest,

/+[out]+/IUnknown\* ppMappedBreakpointRequest)

{

mixin(LogCallMix);

/+

auto bp = ComPtr!(IDebugBreakpointRequest3)(pUserBreakpointRequest);

if(bp)

{

BP\_LOCATION\_TYPE type;

HRESULT hr = bp.GetLocationType(&type);

logCall("type = %x", type);

BP\_REQUEST\_INFO info;

bp.GetRequestInfo(BPREQI\_ALLFIELDS, &info);

if((type & BPLT\_LOCATION\_TYPE\_MASK) == BPLT\_FILE\_LINE)

{

// wrong struct alignment

if(auto dp2 = (&info.bpLocation.bplocCodeFileLine.pDocPos)[1])

{

ScopedBSTR bstrFileName;

dp2.GetFileName(&bstrFileName.bstr);

logCall("filename = %s", to\_string(bstrFileName));

}

}

BP\_REQUEST\_INFO2 info2;

bp.GetRequestInfo2(BPREQI\_ALLFIELDS, &info2);

}

+/

return S\_FALSE;

}

// IVsProvideColorableItems //////////////////////////////////////

\_\_gshared ColorableItem[] colorableItems;

// delete <VisualStudio-User-Root>\FontAndColors\Cache\{A27B4E24-A735-4D1D-B8E7-9716E1E3D8E0}\Version

// if the list of colorableItems changes

static void shared\_static\_this()

{

colorableItems = [

// The first 6 items in this list MUST be these default items.

newCom!ColorableItem("Keyword", CI\_BLUE, CI\_USERTEXT\_BK),

newCom!ColorableItem("Comment", CI\_DARKGREEN, CI\_USERTEXT\_BK),

newCom!ColorableItem("Identifier", CI\_USERTEXT\_FG, CI\_USERTEXT\_BK),

newCom!ColorableItem("String", CI\_MAROON, CI\_USERTEXT\_BK),

newCom!ColorableItem("Number", CI\_USERTEXT\_FG, CI\_USERTEXT\_BK),

newCom!ColorableItem("Text", CI\_USERTEXT\_FG, CI\_USERTEXT\_BK),

// Visual D specific (must match Lexer.TokenCat)

newCom!ColorableItem("Visual D Operator", CI\_USERTEXT\_FG, CI\_USERTEXT\_BK),

newCom!ColorableItem("Visual D Register", -1, CI\_USERTEXT\_BK, RGB(128, 0, 128)),

newCom!ColorableItem("Visual D Mnemonic", CI\_AQUAMARINE, CI\_USERTEXT\_BK),

newCom!ColorableItem("Visual D Type", -1, CI\_USERTEXT\_BK, RGB(0, 0, 160)),

newCom!ColorableItem("Visual D Predefined Version", -1, CI\_USERTEXT\_BK, RGB(160, 0, 0)),

newCom!ColorableItem("Visual D Disabled Keyword", -1, CI\_USERTEXT\_BK, RGB(128, 160, 224)),

newCom!ColorableItem("Visual D Disabled Comment", -1, CI\_USERTEXT\_BK, RGB(96, 128, 96)),

newCom!ColorableItem("Visual D Disabled Identifier", CI\_DARKGRAY, CI\_USERTEXT\_BK),

newCom!ColorableItem("Visual D Disabled String", -1, CI\_USERTEXT\_BK, RGB(192, 160, 160)),

newCom!ColorableItem("Visual D Disabled Number", CI\_DARKGRAY, CI\_USERTEXT\_BK),

newCom!ColorableItem("Visual D Disabled Text", CI\_DARKGRAY, CI\_USERTEXT\_BK),

newCom!ColorableItem("Visual D Disabled Operator", CI\_DARKGRAY, CI\_USERTEXT\_BK),

newCom!ColorableItem("Visual D Disabled Register", -1, CI\_USERTEXT\_BK, RGB(128, 160, 224)),

newCom!ColorableItem("Visual D Disabled Mnemonic", -1, CI\_USERTEXT\_BK, RGB(128, 160, 224)),

newCom!ColorableItem("Visual D Disabled Type", -1, CI\_USERTEXT\_BK, RGB(64, 112, 208)),

newCom!ColorableItem("Visual D Disabled Version", -1, CI\_USERTEXT\_BK, RGB(160, 128, 128)),

newCom!ColorableItem("Visual D Token String Keyword", -1, CI\_USERTEXT\_BK, RGB(160,0,128)),

newCom!ColorableItem("Visual D Token String Comment", -1, CI\_USERTEXT\_BK, RGB(128,160,80)),

newCom!ColorableItem("Visual D Token String Identifier", -1, CI\_USERTEXT\_BK, RGB(128,32,32)),

newCom!ColorableItem("Visual D Token String String", -1, CI\_USERTEXT\_BK, RGB(255,64,64)),

newCom!ColorableItem("Visual D Token String Number", -1, CI\_USERTEXT\_BK, RGB(128,32,32)),

newCom!ColorableItem("Visual D Token String Text", -1, CI\_USERTEXT\_BK, RGB(128,32,32)),

newCom!ColorableItem("Visual D Token String Operator", -1, CI\_USERTEXT\_BK, RGB(128,32,32)),

newCom!ColorableItem("Visual D Token String Register", -1, CI\_USERTEXT\_BK, RGB(192,0,128)),

newCom!ColorableItem("Visual D Token String Mnemonic", -1, CI\_USERTEXT\_BK, RGB(192,0,128)),

newCom!ColorableItem("Visual D Token String Type", -1, CI\_USERTEXT\_BK, RGB(112,0,80)),

newCom!ColorableItem("Visual D Token String Version", -1, CI\_USERTEXT\_BK, RGB(224, 0, 0)),

newCom!ColorableItem("Visual D Text Coverage", CI\_USERTEXT\_FG, -1, 0, RGB(192, 255, 192)),

newCom!ColorableItem("Visual D Text Non-Coverage", CI\_USERTEXT\_FG, -1, 0, RGB(255, 192, 192)),

newCom!ColorableItem("Visual D Margin No Coverage", CI\_USERTEXT\_FG, -1, 0, RGB(192, 192, 192)),

];

};

static void shared\_static\_dtor()

{

destroy(colorableItems); // to keep GC leak detection happy

Source.parseTaskPool = null;

}

static void updateThemeColors()

{

bool dark = Package.GetGlobalOptions().isDarkTheme();

foreach(ci; colorableItems)

{

if(ci.GetDisplayName() == "Visual D Type")

ci.SetDefaultForegroundColor(dark ? RGB(128, 128, 160) : RGB(0, 0, 160));

if(ci.GetDisplayName() == "Visual D Register")

ci.SetDefaultForegroundColor(dark ? RGB(128, 64, 128) : RGB(128, 0, 128));

if(ci.GetDisplayName() == "Visual D Token String Identifier")

ci.SetDefaultForegroundColor(dark ? RGB(128, 64, 64) : RGB(128,32,32));

if(ci.GetDisplayName() == "Visual D Token String Number")

ci.SetDefaultForegroundColor(dark ? RGB(128, 64, 64) : RGB(128,32,32));

if(ci.GetDisplayName() == "Visual D Token String Operator")

ci.SetDefaultForegroundColor(dark ? RGB(128, 64, 64) : RGB(128,32,32));

if(ci.GetDisplayName() == "Visual D Token String Type")

ci.SetDefaultForegroundColor(dark ? RGB(160, 128, 160) : RGB(112, 0, 80));

}

version(none)

{

// only resets user colors?

IVsTextManager2 textmgr = queryService!(VsTextManager, IVsTextManager2);

if(textmgr)

textmgr.ResetColorableItems(g\_languageCLSID);

release(textmgr);

}

}

override HRESULT GetColorableItem(in int iIndex, IVsColorableItem\* ppItem)

{

if(iIndex < 1 || iIndex > colorableItems.length)

return E\_INVALIDARG;

\*ppItem = addref(colorableItems[iIndex-1]);

return S\_OK;

}

override HRESULT GetItemCount(int\* piCount)

{

\*piCount = colorableItems.length;

return S\_OK;

}

// IVsLanguageContextProvider //////////////////////////////////////

override HRESULT UpdateLanguageContext(uint dwHint, IVsTextLines pBuffer, TextSpan\* ptsSelection, IVsUserContext pUC)

{

mixin(LogCallMix);

return E\_NOTIMPL;

}

// IServiceProvider //////////////////////////////////////

override HRESULT QueryService(in GUID\* guidService, in IID\* riid, void \*\* ppvObject)

{

mixin(LogCallMix);

return E\_NOTIMPL;

}

// IVsDebuggerEvents //////////////////////////////////////

override HRESULT OnModeChange(in DBGMODE dbgmodeNew)

{

mixin(LogCallMix2);

mDbgMode = dbgmodeNew;

return S\_OK;

}

// IVsFormatFilterProvider //////////////////////////////////////

override HRESULT CurFileExtensionFormat(in BSTR bstrFileName, uint\* pdwExtnIndex)

{

mixin(LogCallMix2);

return E\_NOTIMPL;

}

override HRESULT GetFormatFilterList(BSTR\* pbstrFilterList)

{

mixin(LogCallMix);

return E\_NOTIMPL;

}

override HRESULT QueryInvalidEncoding(in uint Format, BSTR\* pbstrMessage)

{

mixin(LogCallMix2);

return E\_NOTIMPL;

}

// IVsUpdateSolutionEvents ///////////////////////////////////

HRESULT UpdateSolution\_Begin(/+[in, out]+/ BOOL \*pfCancelUpdate)

{

if(pfCancelUpdate)

\*pfCancelUpdate = false;

return S\_OK;

}

HRESULT UpdateSolution\_Done(in BOOL fSucceeded, in BOOL fModified, in BOOL fCancelCommand)

{

return S\_OK;

}

HRESULT UpdateSolution\_StartUpdate( /+[in, out]+/ BOOL \*pfCancelUpdate )

{

if(pfCancelUpdate)

\*pfCancelUpdate = false;

return S\_OK;

}

HRESULT UpdateSolution\_Cancel()

{

return S\_OK;

}

HRESULT OnActiveProjectCfgChange(/+[in]+/ IVsHierarchy pIVsHierarchy)

{

UpdateColorizer(false);

return S\_OK;

}

void UpdateColorizer(bool force)

{

foreach(src; mSources)

src.mColorizer.OnConfigModified(force);

}

// IVsOutliningCapableLanguage ///////////////////////////////

HRESULT CollapseToDefinitions(/+[in]+/ IVsTextLines pTextLines, // the buffer in question

/+[in]+/ IVsOutliningSession pSession)

{

GetSource(pTextLines).mOutlining = true;

if(auto session = qi\_cast!IVsHiddenTextSession(pSession))

{

scope(exit) release(session);

GetSource(pTextLines).UpdateOutlining(session, hrsDefault);

GetSource(pTextLines).CollapseAllHiddenRegions(session, true);

}

return S\_OK;

}

//////////////////////////////////////////////////////////////

private Source cdwLastSource;

private int cdwLastLine, cdwLastColumn;

public ViewFilter mLastActiveView;

private SysTime mTimeOutSamePos;

bool tryJumpToDefinitionInCodeWindow(Source src, int line, int col)

{

SysTime now = Clock.currTime();

if (cdwLastSource == src && cdwLastLine == line && cdwLastColumn == col)

{

// wait for the caret staying on the same position for a second

if(mTimeOutSamePos > now)

return false;

mTimeOutSamePos += dur!"days"(1);

}

else

{

cdwLastSource = src;

cdwLastLine = line;

cdwLastColumn = col;

mTimeOutSamePos = now + dur!"seconds"(1);

return false;

}

if (src.mDisasmFile.length)

{

int asmline = src.getLineInDisasm(line);

if (asmline < 0)

return false;

return jumpToDefinitionInCodeWindow("", src.mDisasmFile, asmline, 0, false);

}

int startIdx, endIdx;

if(!src.GetWordExtent(line, col, WORDEXT\_CURRENT, startIdx, endIdx))

return false;

string word = toUTF8(src.GetText(line, startIdx, line, endIdx));

if(word.length <= 0)

return false;

Definition[] defs = Package.GetLibInfos().findDefinition(word);

if(defs.length == 0)

return false;

string srcfile = src.GetFileName();

string abspath;

if(FindFileInSolution(defs[0].filename, srcfile, abspath) != S\_OK)

return false;

return jumpToDefinitionInCodeWindow("", abspath, defs[0].line, 0, false);

}

//////////////////////////////////////////////////////////////

bool mGCdisabled;

SysTime mLastExecTime;

size\_t mGCUsedSize;

enum PAGESIZE = 4096;

void OnExec()

{

version(TWEAK\_GC)

if(false && !mGCdisabled)

{

GC.disable();

mGCdisabled = true;

//auto stats = gc\_stats();

//mGCUsedSize = stats.usedsize + PAGESIZE \* stats.pageblocks;

}

mLastExecTime = Clock.currTime() + dur!"seconds"(2);

}

void CheckGC(bool forceEnable)

{

if(!mGCdisabled)

return;

SysTime now = Clock.currTime();

version(TWEAK\_GC)

if(forceEnable || mLastExecTime < now)

{

GC.enable();

auto stats = gc\_stats();

auto usedSize = stats.usedsize + PAGESIZE \* stats.pageblocks;

if(usedSize > mGCUsedSize + (20<<20))

{

GC.collect();

stats = gc\_stats();

mGCUsedSize = stats.usedsize + PAGESIZE \* stats.pageblocks;

}

mGCdisabled = false;

}

}

//////////////////////////////////////////////////////////////

bool OnIdle()

{

if(mVDServerClient)

mVDServerClient.onIdle();

CheckGC(false);

for(int i = 0; i < mSources.length; i++)

if(mSources[i].OnIdle())

return true;

foreach(CodeWindowManager mgr; mCodeWinMgrs)

if(mgr.OnIdle())

return true;

if(mLastActiveView && mLastActiveView.mView)

{

int line, idx;

mLastActiveView.mView.GetCaretPos(&line, &idx);

if(tryJumpToDefinitionInCodeWindow(mLastActiveView.mCodeWinMgr.mSource, line, idx))

return true;

}

return false;

}

Source GetSource(IVsTextLines buffer, bool create = true)

{

Source src;

for(int i = 0; i < mSources.length; i++)

{

src = mSources[i];

if(src.mBuffer is buffer)

goto L\_found;

}

if(!create)

return null;

src = newCom!Source(buffer);

mSources ~= src;

src.AddRef();

L\_found:

return src;

}

Source GetSource(string filename)

{

for(int i = 0; i < mSources.length; i++)

{

string srcfile = mSources[i].GetFileName();

if(CompareFilenames(srcfile, filename) == 0)

return mSources[i];

}

return null;

}

Source[] GetSources()

{

return mSources;

}

IVsTextView GetView(string filename)

{

foreach(cmgr; mCodeWinMgrs)

{

string srcfile = cmgr.mSource.GetFileName();

if(CompareFilenames(srcfile, filename) == 0)

{

if (cmgr.mViewFilters.length)

return cmgr.mViewFilters[0].mView;

return null;

}

}

return null;

}

void setDebugger(IVsDebugger debugger)

{

if(mCookieDebuggerEvents && mDebugger)

{

mDebugger.UnadviseDebuggerEvents(mCookieDebuggerEvents);

mCookieDebuggerEvents = 0;

}

mDebugger = release(mDebugger);

mDebugger = addref(debugger);

if(mDebugger)

mDebugger.AdviseDebuggerEvents(this, &mCookieDebuggerEvents);

}

bool IsDebugging()

{

return (mDbgMode & ~ DBGMODE\_EncMask) != DBGMODE\_Design;

}

bool GetCoverageData(string filename, uint line, uint\* data, uint cnt, float\* covPrecent)

{

if(!Package.GetGlobalOptions().showCoverageMargin)

return false;

Source src = GetSource(filename);

if(!src)

return false;

auto cov = src.mColorizer.mCoverage;

if(cov.length == 0)

return false;

for(uint ln = 0; ln < cnt; ln++)

{

uint covLine = src.adjustLineNumberSinceLastBuildReverse(line + ln, true);

data[ln] = covLine >= cov.length ? -1 : cov[covLine];

}

if (covPrecent)

\*covPrecent = src.mColorizer.mCoveragePercent;

return true;

}

// semantic completion ///////////////////////////////////

uint GetTip(Source src, TextSpan\* pSpan, GetTipCallBack cb)

{

ConfigureSemanticProject(src);

return vdServerClient.GetTip(src.GetFileName(), pSpan, cb);

}

uint GetDefinition(Source src, TextSpan\* pSpan, GetDefinitionCallBack cb)

{

ConfigureSemanticProject(src);

return vdServerClient.GetDefinition(src.GetFileName(), pSpan, cb);

}

uint GetSemanticExpansions(Source src, string tok, int line, int idx, GetExpansionsCallBack cb)

{

ConfigureSemanticProject(src);

wstring expr = src.FindExpressionBefore(line, idx);

return vdServerClient.GetSemanticExpansions(src.GetFileName(), tok, line, idx, expr, cb);

}

uint GetReferences(Source src, string tok, int line, int idx, GetReferencesCallBack cb)

{

ConfigureSemanticProject(src);

wstring expr;

return vdServerClient.GetReferences(src.GetFileName(), tok, line, idx, expr, cb);

}

void UpdateSemanticModule(Source src)

{

}

void ClearSemanticProject()

{

vdServerClient.ClearSemanticProject();

}

void ConfigureSemanticProject(Source src)

{

string file = src.GetFileName();

string[] imp = GetImportPaths(file);

string[] stringImp;

string[] versionids;

string[] debugids;

uint flags = 0;

Config cfg = getProjectConfig(file);

if(!cfg)

cfg = getCurrentStartupConfig();

if(cfg)

{

scope(exit) release(cfg);

auto cfgopts = cfg.GetProjectOptions();

auto globopts = Package.GetGlobalOptions();

flags = ConfigureFlags!()(cfgopts.useUnitTests, !cfgopts.release, cfgopts.isX86\_64,

cfgopts.cov, cfgopts.doDocComments, cfgopts.noboundscheck,

cfgopts.compiler == Compiler.GDC,

cfgopts.versionlevel, cfgopts.debuglevel,

cfgopts.errDeprecated, cfgopts.compiler == Compiler.LDC,

cfgopts.useMSVCRT (), globopts.mixinAnalysis, globopts.UFCSExpansions);

string strimp = cfgopts.replaceEnvironment(cfgopts.fileImppath, cfg);

stringImp = tokenizeArgs(strimp);

foreach(ref i; stringImp)

i = normalizeDir(unquoteArgument(i));

makeFilenamesAbsolute(stringImp, cfg.GetProjectDir());

versionids = tokenizeArgs(cfgopts.versionids);

debugids = tokenizeArgs(cfgopts.debugids);

}

vdServerClient.ConfigureSemanticProject(file, assumeUnique(imp), assumeUnique(stringImp), assumeUnique(versionids), assumeUnique(debugids), flags);

}

bool isBinaryOperator(Source src, int startLine, int startIndex, int endLine, int endIndex)

{

auto pos = vdc.util.TextPos(startIndex, startLine);

return src.mBinaryIsIn.contains(pos) !is null;

//return vdServerClient.isBinaryOperator(src.GetFileName(), startLine, startIndex, endLine, endIndex);

}

private:

//Package mPackage;

Source[] mSources;

CodeWindowManager[] mCodeWinMgrs;

DBGMODE mDbgMode;

VDServerClient mVDServerClient;

IVsDebugger mDebugger;

VSCOOKIE mCookieDebuggerEvents = VSCOOKIE\_NIL;

VSCOOKIE mUpdateSolutionEventsCookie = VSCOOKIE\_NIL;

UpdateSolutionEvents mUpdateSolutionEvents;

}

///////////////////////////////////////////////////////////////////////////////

// seperate object from LanguageService to avoid circular references

class UpdateSolutionEvents : DComObject, IVsUpdateSolutionEvents

{

LanguageService mLangSvc;

this(LanguageService svc)

{

mLangSvc = svc;

}

override HRESULT QueryInterface(in IID\* riid, void\*\* pvObject)

{

if(queryInterface!(IVsUpdateSolutionEvents) (this, riid, pvObject))

return S\_OK;

return super.QueryInterface(riid, pvObject);

}

// IVsUpdateSolutionEvents ///////////////////////////////////

HRESULT UpdateSolution\_Begin(/+[in, out]+/ BOOL \*pfCancelUpdate)

{

return mLangSvc.UpdateSolution\_Begin(pfCancelUpdate);

}

HRESULT UpdateSolution\_Done(in BOOL fSucceeded, in BOOL fModified, in BOOL fCancelCommand)

{

return mLangSvc.UpdateSolution\_Done(fSucceeded, fModified, fCancelCommand);

}

HRESULT UpdateSolution\_StartUpdate( /+[in, out]+/ BOOL \*pfCancelUpdate )

{

return mLangSvc.UpdateSolution\_StartUpdate(pfCancelUpdate);

}

HRESULT UpdateSolution\_Cancel()

{

return mLangSvc.UpdateSolution\_Cancel();

}

HRESULT OnActiveProjectCfgChange(/+[in]+/ IVsHierarchy pIVsHierarchy)

{

return mLangSvc.OnActiveProjectCfgChange(pIVsHierarchy);

}

}

///////////////////////////////////////////////////////////////////////////////

class CodeWindowManager : DisposingComObject, IVsCodeWindowManager

{

IVsCodeWindow mCodeWin;

Source mSource;

LanguageService mLangSvc;

ViewFilter[] mViewFilters;

this(LanguageService langSvc, IVsCodeWindow pCodeWin, Source source)

{

mCodeWin = pCodeWin;

if(mCodeWin)

{

mCodeWin.AddRef();

}

mSource = addref(source);

mLangSvc = langSvc;

}

~this()

{

}

override void Dispose()

{

CloseFilters();

if(mCodeWin)

{

mCodeWin.Release();

mCodeWin = null;

}

mSource = release(mSource);

mLangSvc = null;

}

override HRESULT QueryInterface(in IID\* riid, void\*\* pvObject)

{

if(queryInterface!(IVsCodeWindowManager) (this, riid, pvObject))

return S\_OK;

return super.QueryInterface(riid, pvObject);

}

// IVsCodeWindowManager //////////////////////////////////////

override int AddAdornments()

{

mixin(LogCallMix);

IVsTextView textView;

if(mCodeWin.GetPrimaryView(&textView) != S\_OK)

return E\_FAIL;

// attach view filter to primary view.

if(textView)

OnNewView(textView);

// attach view filter to secondary view.

textView = null;

if(mCodeWin.GetSecondaryView(&textView) != S\_OK)

return E\_FAIL;

if(textView)

OnNewView(textView);

return S\_OK;

}

override int RemoveAdornments()

{

mixin(LogCallMix);

CloseFilters();

return S\_OK;

}

override int OnNewView(IVsTextView pView)

{

mixin(LogCallMix);

ViewFilter vf = newCom!ViewFilter(this, pView);

mViewFilters ~= vf;

return S\_OK;

}

//////////////////////////////////////////////////////////////////////

bool OnIdle()

{

foreach(ViewFilter vf; mViewFilters)

if(vf.OnIdle())

return true;

return false;

}

void CloseFilters()

{

foreach(ViewFilter vf; mViewFilters)

vf.Dispose();

mViewFilters = mViewFilters.init;

}

ViewFilter GetViewFilter(IVsTextView pView)

{

foreach(vf; mViewFilters)

if(vf.mView is pView)

return vf;

return null;

}

}

/////////////////////////////////////////////////////////////////////////

class CodeDefViewContext : DComObject, IVsCodeDefViewContext

{

private string symbol;

private string filename;

private int line;

private int column;

this(string symbol, string filename, int line, int col)

{

this.symbol = symbol;

this.filename = filename;

this.line = line;

this.column = col;

}

override HRESULT QueryInterface(in IID\* riid, void\*\* pvObject)

{

if(queryInterface!(IVsCodeDefViewContext) (this, riid, pvObject))

return S\_OK;

return super.QueryInterface(riid, pvObject);

}

override HRESULT GetCount(ULONG\* pcItems)

{

\*pcItems = 1;

return S\_OK;

}

override HRESULT GetCol(in ULONG iItem, ULONG\* piCol)

{

\*piCol = column;

return S\_OK;

}

override HRESULT GetLine(in ULONG iItem, ULONG\* piLine)

{

\*piLine = line;

return S\_OK;

}

override HRESULT GetFileName(in ULONG iItem, BSTR \*pbstrFilename)

{

\*pbstrFilename = allocBSTR(filename);

return S\_OK;

}

override HRESULT GetSymbolName(in ULONG iItem, BSTR \*pbstrSymbolName)

{

\*pbstrSymbolName = allocBSTR(symbol);

return S\_OK;

}

}

/////////////////////////////////////////////////////////////////////////

HRESULT reloadTextBuffer(string fname)

{

IVsRunningDocumentTable pRDT = queryService!(IVsRunningDocumentTable);

if(!pRDT)

return E\_FAIL;

scope(exit) release(pRDT);

auto docname = \_toUTF16z(fname);

IVsHierarchy srpIVsHierarchy;

VSITEMID vsItemId = VSITEMID\_NIL;

IUnknown srpIUnknown;

VSDOCCOOKIE vsDocCookie = VSDOCCOOKIE\_NIL;

HRESULT hr = pRDT.FindAndLockDocument(/\* [in] VSRDTFLAGS dwRDTLockType \*/ RDT\_NoLock,

/\* [in] LPCOLESTR pszMkDocument \*/ docname,

/\* [out] IVsHierarchy \*\*ppHier \*/ &srpIVsHierarchy,

/\* [out] VSITEMID \*pitemid \*/ &vsItemId,

/\* [out] IUnknown \*\*ppunkDocData \*/ &srpIUnknown,

/\* [out] VSCOOKIE \*pdwCookie \*/ &vsDocCookie);

// FindAndLockDocument returns S\_FALSE if the doc is not in the RDT

if (hr != S\_OK)

return hr;

scope(exit) release(srpIUnknown);

scope(exit) release(srpIVsHierarchy);

IVsTextLines textBuffer = qi\_cast!IVsTextLines(srpIUnknown);

if(!textBuffer)

if(auto bufferProvider = qi\_cast!IVsTextBufferProvider(srpIUnknown))

{

bufferProvider.GetTextBuffer(&textBuffer);

release(bufferProvider);

}

if(!textBuffer)

return returnError(E\_FAIL);

scope(exit) release(textBuffer);

if (auto docdata = qi\_cast!IVsPersistDocData(srpIUnknown))

docdata.ReloadDocData(RDD\_IgnoreNextFileChange|RDD\_RemoveUndoStack);

return textBuffer.Reload(true);

}

IVsTextView findCodeDefinitionWindow()

{

IVsCodeDefView cdv = queryService!(SVsCodeDefView,IVsCodeDefView);

if (!cdv)

return null;

scope(exit) release(cdv);

IVsTextManager textmgr = queryService!(VsTextManager, IVsTextManager);

if(!textmgr)

return null;

scope(exit) release(textmgr);

IVsEnumTextViews enumTextViews;

// Passing null will return all available views, at least according to the documentation

// unfortunately, it returns error E\_INVALIDARG, said to be not implemented

HRESULT hr = textmgr.EnumViews(null, &enumTextViews);

if (!enumTextViews)

return null;

scope(exit) release(enumTextViews);

IVsTextView tv;

DWORD fetched;

while(enumTextViews.Next(1, &tv, &fetched) == S\_OK && fetched == 1)

{

BOOL result;

if (cdv.IsCodeDefView(tv, &result) == S\_OK && result)

return tv;

}

return null;

}

bool jumpToDefinitionInCodeWindow(string symbol, string filename, int line, int col, bool forceShow)

{

IVsCodeDefView cdv = queryService!(SVsCodeDefView,IVsCodeDefView);

if (cdv is null)

return false;

scope(exit) release(cdv);

if (!forceShow && cdv.IsVisible() != S\_OK)

return false;

CodeDefViewContext context = newCom!CodeDefViewContext(symbol, filename, line, col);

cdv.SetContext(context);

if (forceShow)

{

if (cdv.IsVisible() != S\_OK)

cdv.ShowWindow();

cdv.ForceIdleProcessing();

}

return true;

}

///////////////////////////////////////////////////////////////////////////////

int GetUserPreferences(LANGPREFERENCES \*langPrefs, IVsTextView view)

{

IVsTextManager textmgr = queryService!(VsTextManager, IVsTextManager);

if(!textmgr)

return E\_FAIL;

scope(exit) release(textmgr);

langPrefs.guidLang = g\_languageCLSID;

if(int rc = textmgr.GetUserPreferences(null, null, langPrefs, null))

return rc;

if (view)

{

int flags, tabsize, indentsize;

if(vdhelper\_GetTextOptions(view, &flags, &tabsize, &indentsize) == S\_OK)

{

langPrefs.uTabSize = max(1, tabsize);

langPrefs.uIndentSize = max(1, indentsize);

langPrefs.fInsertTabs = (flags & 1) == 0;

}

}

return S\_OK;

}

// An object to break cyclic dependencies on Source

class SourceEvents : DisposingComObject, IVsUserDataEvents, IVsTextLinesEvents

{

Source mSource;

uint mCookieUserDataEvents;

uint mCookieTextLinesEvents;

this(Source src, IVsTextLines buffer)

{

mSource = src;

if(buffer)

{

mCookieUserDataEvents = Advise!(IVsUserDataEvents)(buffer, this);

mCookieTextLinesEvents = Advise!(IVsTextLinesEvents)(buffer, this);

}

}

override void Dispose()

{

IVsTextLines buffer = mSource.mBuffer;

if(buffer)

{

if(mCookieUserDataEvents)

Unadvise!(IVsUserDataEvents)(buffer, mCookieUserDataEvents);

if(mCookieTextLinesEvents)

Unadvise!(IVsTextLinesEvents)(buffer, mCookieTextLinesEvents);

}

}

override HRESULT QueryInterface(in IID\* riid, void\*\* pvObject)

{

if(queryInterface!(IVsUserDataEvents) (this, riid, pvObject))

return S\_OK;

if(queryInterface!(IVsTextLinesEvents) (this, riid, pvObject))

return S\_OK;

return super.QueryInterface(riid, pvObject);

}

// IVsUserDataEvents //////////////////////////////////////

override int OnUserDataChange(in GUID\* riidKey, in VARIANT vtNewValue)

{

return mSource.OnUserDataChange(riidKey, vtNewValue);

}

// IVsTextLinesEvents //////////////////////////////////////

override int OnChangeLineText(in TextLineChange \*pTextLineChange, in BOOL fLast)

{

return mSource.OnChangeLineText(pTextLineChange, fLast);

}

override int OnChangeLineAttributes(in int iFirstLine, in int iLastLine)

{

return mSource.OnChangeLineAttributes(iFirstLine, iLastLine);

}

}

struct ParseError

{

ParserSpan span;

string msg;

}

class Source : DisposingComObject, IVsUserDataEvents, IVsTextLinesEvents, IVsTextMarkerClient

{

Colorizer mColorizer;

IVsTextLines mBuffer;

CompletionSet mCompletionSet;

MethodData mMethodData;

ExpansionProvider mExpansionProvider;

SourceEvents mSourceEvents;

bool mOutlining;

bool mStopOutlining;

bool mVerifiedEncoding;

IVsHiddenTextSession mHiddenTextSession;

static struct LineChange { int oldLine, newLine; }

LineChange[] mLineChanges;

size\_t mLastSaveLineChangePos;

TextLineChange mLastTextLineChange;

wstring mParseText;

ParseError[] mParseErrors;

vdc.util.TextPos[] mBinaryIsIn;

NewHiddenRegion[] mOutlineRegions;

int mParsingState;

int mModificationCountAST;

int mModificationCount;

string mDisasmFile;

string mLineInfoFile;

LineInfo[] mDisasmLineInfo;

SymLineInfo[string] mDisasmSymInfo;

this(IVsTextLines buffer)

{

mBuffer = addref(buffer);

mColorizer = newCom!Colorizer(this);

mSourceEvents = newCom!SourceEvents(this, mBuffer);

mOutlining = Package.GetGlobalOptions().autoOutlining;

mModificationCountAST = -1;

}

~this()

{

}

override void Dispose()

{

mExpansionProvider = release(mExpansionProvider);

DismissCompletor();

DismissMethodTip();

mCompletionSet = release(mCompletionSet);

if(mMethodData)

{

mMethodData.Dispose(); // we need to break the circular reference MethodData<->IVsMethodTipWindow

mMethodData = release(mMethodData);

}

mSourceEvents.Dispose();

mSourceEvents = null;

mBuffer = release(mBuffer);

mHiddenTextSession = release(mHiddenTextSession);

mColorizer = null;

}

override HRESULT QueryInterface(in IID\* riid, void\*\* pvObject)

{

if(queryInterface!(IVsUserDataEvents) (this, riid, pvObject))

return S\_OK;

if(queryInterface!(IVsTextLinesEvents) (this, riid, pvObject))

return S\_OK;

if(queryInterface!(IVsTextMarkerClient) (this, riid, pvObject))

return S\_OK;

return super.QueryInterface(riid, pvObject);

}

void setUtf8Encoding()

{

if(auto ud = qi\_cast!IVsUserData(mBuffer))

{

scope(exit) release(ud);

//object oname;

//Guid GUID\_VsBufferMoniker = typeof(IVsUserData).GUID;

VARIANT var;

if(SUCCEEDED(ud.GetData(&GUID\_VsBufferEncodingVSTFF, &var)))

{

uint dwBufferVSTFF = var.ulVal;

uint codepage = dwBufferVSTFF & VSTFF\_CPMASK; // to extract codepage

uint vstffFlags = dwBufferVSTFF & VSTFF\_FLAGSMASK; // to extract CHARFMT

if(!(vstffFlags & VSTFF\_SIGNATURE) && codepage != 65001) // no signature, and not utf8

{

var.ulVal = vstffFlags | 65001;

ud.SetData(&GUID\_VsBufferEncodingVSTFF, var);

}

}

}

}

// IVsUserDataEvents //////////////////////////////////////

override int OnUserDataChange(in GUID\* riidKey, in VARIANT vtNewValue)

{

return S\_OK;

}

// IVsTextLinesEvents //////////////////////////////////////

override int OnChangeLineText(in TextLineChange \*pTextLineChange, in BOOL fLast)

{

mLastTextLineChange = \*pTextLineChange;

mModificationCount++;

if(!mVerifiedEncoding)

{

mVerifiedEncoding = true;

setUtf8Encoding();

}

if(pTextLineChange.iOldEndLine != pTextLineChange.iNewEndLine)

{

bool skip = false;

if(pTextLineChange.iStartLine == 0 && pTextLineChange.iOldEndLine == 0)

{

// is this the first insert that actually fills the Source with the file content?

skip = (GetLineCount() == pTextLineChange.iNewEndLine + 1);

}

if(!skip)

{

LineChange chg = LineChange(pTextLineChange.iOldEndLine, pTextLineChange.iNewEndLine);

mLineChanges ~= chg;

}

}

if(mOutlining)

CheckOutlining(pTextLineChange);

return mColorizer.OnLinesChanged(pTextLineChange.iStartLine, pTextLineChange.iOldEndLine, pTextLineChange.iNewEndLine, fLast != 0);

}

void ClearLineChanges()

{

mLineChanges = mLineChanges.init;

mLastSaveLineChangePos = 0;

}

override int OnChangeLineAttributes(in int iFirstLine, in int iLastLine)

{

return S\_OK;

}

HRESULT ReColorizeLines(int iTopLine, int iBottomLine)

{

if(IVsTextColorState colorState = qi\_cast!IVsTextColorState(mBuffer))

{

scope(exit) release(colorState);

if(iBottomLine == -1)

iBottomLine = GetLineCount() - 1;

colorState.ReColorizeLines(iTopLine, iBottomLine);

}

return S\_OK;

}

int adjustLineNumberSinceLastBuild(int line, bool sinceSave)

{

size\_t pos = sinceSave ? mLastSaveLineChangePos : 0;

foreach(ref chg; mLineChanges[pos..$])

if(line >= chg.oldLine)

line += chg.newLine - chg.oldLine;

return line;

}

int adjustLineNumberSinceLastBuildReverse(int line, bool sinceSave)

{

size\_t pos = sinceSave ? mLastSaveLineChangePos : 0;

foreach\_reverse(ref chg; mLineChanges[pos..$])

if(line >= chg.newLine)

line -= chg.newLine - chg.oldLine;

return line;

}

// IVsTextMarkerClient //////////////////////////////////////

override HRESULT MarkerInvalidated()

{

return S\_OK;

}

override HRESULT GetTipText(/+[in]+/ IVsTextMarker pMarker,

/+[out, optional]+/ BSTR \*pbstrText)

{

if(auto marker = qi\_cast!IVsTextLineMarker(pMarker))

{

scope(exit) marker.Release();

TextSpan span;

if(marker.GetCurrentSpan(&span) == S\_OK)

{

string tip = getParseError(span.iStartLine, span.iStartIndex);

if(tip.length)

{

\*pbstrText = allocBSTR(tip);

return S\_OK;

}

}

}

return E\_FAIL;

}

override HRESULT OnBufferSave(LPCOLESTR pszFileName)

{

mLastSaveLineChangePos = mLineChanges.length;

return S\_OK;

}

override HRESULT OnBeforeBufferClose()

{

return S\_OK;

}

// Commands -- see MarkerCommandValues for meaning of iItem param

override HRESULT GetMarkerCommandInfo(/+[in]+/ IVsTextMarker pMarker, in int iItem,

/+[out, custom(uuid\_IVsTextMarkerClient, "optional")]+/ BSTR \* pbstrText,

/+[out]+/ DWORD\* pcmdf)

{

return E\_NOTIMPL;

}

override HRESULT ExecMarkerCommand(/+[in]+/ IVsTextMarker pMarker, in int iItem)

{

return E\_NOTIMPL;

}

override HRESULT OnAfterSpanReload()

{

return S\_OK;

}

override HRESULT OnAfterMarkerChange(/+[in]+/ IVsTextMarker pMarker)

{

return S\_OK;

}

///////////////////////////////////////////////////////////////////////////////

void setDisasmFiles(string asmfile, string linefile)

{

mDisasmFile = asmfile;

mLineInfoFile = linefile;

try

{

GlobalOptions globOpt = Package.GetGlobalOptions();

if(globOpt.demangleError)

asmfile ~= ".mangled";

mDisasmSymInfo = readDisasmFile(asmfile);

mDisasmLineInfo = readLineInfoFile(linefile, GetFileName());

// force update to Code Definition Window

auto langsvc = Package.GetLanguageService();

int line, idx;

if (langsvc.mLastActiveView && langsvc.mLastActiveView.mView &&

langsvc.mLastActiveView.mCodeWinMgr.mSource == this)

langsvc.mLastActiveView.mView.GetCaretPos(&line, &idx);

reloadTextBuffer(mDisasmFile);

int asmline = getLineInDisasm(line);

jumpToDefinitionInCodeWindow("", mDisasmFile, asmline, 0, true);

}

catch(Exception e)

{

writeToBuildOutputPane(e.msg);

}

}

int getLineInDisasm(int line)

{

line++; // 0-based line numbers in VS to 1-based line numbers in debug info

if (line >= mDisasmLineInfo.length)

line = mDisasmLineInfo.length - 1;

// prefer to display asm of line before current line if none available on it

while (line > 0 && mDisasmLineInfo[line].sym is null)

line--;

// fall back to display asm of first line in the file

while (line < mDisasmLineInfo.length && mDisasmLineInfo[line].sym is null)

line++;

if (line >= mDisasmLineInfo.length)

return -1;

SymLineInfo\* symInfo = mDisasmLineInfo[line].sym in mDisasmSymInfo;

if (!symInfo)

return -1;

foreach (i, off; symInfo.offsets)

if (off >= mDisasmLineInfo[line].offset)

return symInfo.firstLine + i;

return -1;

}

///////////////////////////////////////////////////////////////////////////////

enum

{

kOutlineStateValid,

kOutlineStateDirty,

kOutlineStateDirtyIdle,

kOutlineStateDirtyIdle2,

}

int mOutlineState = kOutlineStateDirty;

bool OnIdle()

{

if(mColorizer.UpdateCoverage(false))

return true;

if(startParsing())

return true;

version(threadedOutlining)

{

return false;

} else {

if(!mOutlining)

return false;

final switch(mOutlineState)

{

case kOutlineStateDirtyIdle2:

UpdateOutlining();

mOutlineState = kOutlineStateValid;

return true;

case kOutlineStateDirty:

mOutlineState = kOutlineStateDirtyIdle;

return false;

case kOutlineStateDirtyIdle:

mOutlineState = kOutlineStateDirtyIdle2;

return false;

case kOutlineStateValid:

return false;

}

}

}

void CheckOutlining(in TextLineChange \*pTextLineChange)

{

version(threadedOutlining) {} else

mOutlineState = kOutlineStateDirty;

}

IVsHiddenTextSession GetHiddenTextSession()

{

if(mHiddenTextSession)

return mHiddenTextSession;

if(auto htm = queryService!(VsTextManager, IVsHiddenTextManager))

{

scope(exit) release(htm);

if(htm.GetHiddenTextSession(mBuffer, &mHiddenTextSession) != S\_OK)

htm.CreateHiddenTextSession(0, mBuffer, null, &mHiddenTextSession);

}

return mHiddenTextSession;

}

enum int kHiddenRegionCookie = 37;

bool AnyOutlineExpanded(IVsHiddenTextSession session)

{

IVsEnumHiddenRegions penum;

TextSpan span = TextSpan(0, 0, 0, GetLineCount());

session.EnumHiddenRegions(FHR\_BY\_CLIENT\_DATA, kHiddenRegionCookie, &span, &penum);

IVsHiddenRegion region;

uint fetched;

int hiddenLine = -1;

bool expanded = false;

while (!expanded && penum.Next(1, &region, &fetched) == S\_OK && fetched == 1)

{

uint state;

region.GetState(&state);

region.GetSpan(&span);

release(region);

if(span.iStartLine <= hiddenLine)

continue;

if(state == hrsExpanded)

expanded = true;

hiddenLine = span.iEndLine;

}

release(penum);

return expanded;

}

void UpdateOutlining()

{

if(auto session = GetHiddenTextSession())

UpdateOutlining(session, hrsExpanded);

}

HRESULT StopOutlining()

{

if(mOutlining)

{

mStopOutlining = true;

version(threadedOutlining)

mModificationCount++; // trigger reparsing

else

CheckOutlining(null);

}

return S\_OK;

}

HRESULT ToggleOutlining()

{

if(mOutlining)

{

if(auto session = GetHiddenTextSession())

CollapseAllHiddenRegions(session, AnyOutlineExpanded(session));

}

return S\_OK;

}

void UpdateOutlining(IVsHiddenTextSession session, int state)

{

NewHiddenRegion[] rgns = CreateOutlineRegions(state);

if(DiffRegions(session, rgns))

session.AddHiddenRegions(chrNonUndoable, rgns.length, rgns.ptr, null);

}

void CollapseAllHiddenRegions(IVsHiddenTextSession session, bool collapsed)

{

IVsEnumHiddenRegions penum;

TextSpan span = TextSpan(0, 0, 0, GetLineCount());

session.EnumHiddenRegions(FHR\_BY\_CLIENT\_DATA, kHiddenRegionCookie, &span, &penum);

IVsHiddenRegion region;

uint fetched;

while (penum.Next(1, &region, &fetched) == S\_OK && fetched == 1)

{

region.SetState(collapsed ? hrsDefault : hrsExpanded, chrDefault);

release(region);

}

release(penum);

}

NewHiddenRegion[] CreateOutlineRegions(int expansionState)

{

wstring source = GetText(); // should not be read from another thread

return CreateOutlineRegions(source, expansionState);

}

NewHiddenRegion[] CreateOutlineRegions(wstring source, int expansionState)

{

NewHiddenRegion[] rgns;

int lastOpenRegion = -1; // builds chain with iEndIndex of TextSpan

Lexer lex;

int state = 0;

int lastCommentStartLine = -1;

int lastCommentStartLineLength = 0;

int prevLineLenth = 0;

int ln = 0;

int prevBracketLine = -1;

foreach(txt; splitter(source, '\n'))

{

if(mModificationCountAST != mModificationCount)

break;

//wstring txt = GetText(ln, 0, ln, -1);

if(txt.length > 0 && txt[$-1] == '\r')

txt = txt[0..$-1];

uint pos = 0;

bool isSpaceOrComment = true;

bool isComment = false;

while(pos < txt.length)

{

uint prevpos = pos;

int col = dLex.scan(state, txt, pos);

if(col == TokenCat.Operator)

{

if(txt[pos-1] == '{' || txt[pos-1] == '[')

{

NewHiddenRegion rgn;

rgn.iType = hrtCollapsible;

rgn.dwBehavior = hrbClientControlled;

rgn.dwState = expansionState;

if(ln > prevBracketLine+1 && isSpaceOrComment && !isComment) // move into previous line

rgn.tsHiddenText = TextSpan(prevLineLenth, ln-1, lastOpenRegion, -1);

else

rgn.tsHiddenText = TextSpan(pos - 1, ln, lastOpenRegion, -1);

rgn.pszBanner = txt[pos-1] == '{' ? "{...}"w.ptr : "[...]"w.ptr;

rgn.dwClient = kHiddenRegionCookie;

lastOpenRegion = rgns.length;

rgns ~= rgn;

prevBracketLine = ln;

}

else if((txt[pos-1] == '}' || txt[pos-1] == ']') && lastOpenRegion >= 0)

{

int idx = lastOpenRegion;

lastOpenRegion = rgns[idx].tsHiddenText.iEndIndex;

if(rgns[idx].tsHiddenText.iStartLine == ln)

{

for(int i = idx; i < rgns.length - 1; i++)

rgns[i] = rgns[i + 1];

rgns.length = rgns.length - 1;

}

else

{

rgns[idx].tsHiddenText.iEndIndex = pos;

rgns[idx].tsHiddenText.iEndLine = ln;

}

prevBracketLine = ln;

}

}

isComment = isComment || (col == TokenCat.Comment);

isSpaceOrComment = isSpaceOrComment && Lexer.isCommentOrSpace(col, txt[prevpos .. pos]);

}

if(lastCommentStartLine >= 0)

{

// do not fold single comment line with subsequent empty line

if(!isSpaceOrComment || (!isComment && lastCommentStartLine + 1 == ln))

{

if(lastCommentStartLine + 1 < ln)

{

NewHiddenRegion rgn;

rgn.iType = hrtCollapsible;

rgn.dwBehavior = hrbClientControlled;

rgn.dwState = expansionState;

rgn.tsHiddenText = TextSpan(lastCommentStartLineLength, lastCommentStartLine, prevLineLenth, ln - 1);

rgn.pszBanner = "..."w.ptr;

rgn.dwClient = kHiddenRegionCookie;

rgns ~= rgn;

}

lastCommentStartLine = -1;

}

}

else if(isComment && isSpaceOrComment)

{

lastCommentStartLine = ln;

lastCommentStartLineLength = txt.length;

}

prevLineLenth = txt.length;

ln++;

}

while(lastOpenRegion >= 0)

{

int idx = lastOpenRegion;

lastOpenRegion = rgns[idx].tsHiddenText.iEndIndex;

rgns[idx].tsHiddenText.iEndIndex = 0;

rgns[idx].tsHiddenText.iEndLine = ln;

rgns[idx].pszBanner = rgns[idx].pszBanner[0] == '{' ? "{..."w.ptr : "[..."w.ptr;

}

return rgns;

}

version(none) unittest

{

const(void)\* p = typeid(NewHiddenRegion).rtInfo;

assert(p !is rtinfoNoPointers && p !is rtinfoHasPointers);

}

bool DiffRegions(IVsHiddenTextSession session, ref NewHiddenRegion[] rgns)

{

// Compare the existing regions with the new regions and

// remove any that do not match the new regions.

IVsEnumHiddenRegions penum;

TextSpan span = TextSpan(0, 0, 0, GetLineCount());

session.EnumHiddenRegions(FHR\_BY\_CLIENT\_DATA, kHiddenRegionCookie, &span, &penum);

uint found = 0;

uint enumerated = 0;

uint fetched;

IVsHiddenRegion region;

while(penum.Next(1, &region, &fetched) == S\_OK && fetched == 1)

{

enumerated++;

region.GetSpan(&span);

int i;

for(i = 0; i < rgns.length; i++)

if(rgns[i].tsHiddenText == span)

break;

if(i < rgns.length)

{

for(int j = i + 1; j < rgns.length; j++)

rgns[j-1] = rgns[j];

rgns.length = rgns.length - 1;

found++;

}

else

region.Invalidate(chrNonUndoable);

release(region);

}

release(penum);

// validate regions against current text

int lines = GetLineCount();

for(int i = 0; i < rgns.length; i++)

{

with(rgns[i].tsHiddenText)

{

if(iStartLine >= lines)

{

rgns.length = i;

break;

}

if(iEndLine >= lines)

iEndLine = lines;

int length;

mBuffer.GetLengthOfLine(iStartLine, &length);

if(iStartIndex >= length)

iStartIndex = length;

if(iStartLine != iEndLine)

mBuffer.GetLengthOfLine(iEndLine, &length);

if(iEndIndex >= length)

iEndIndex = length;

}

}

return found != enumerated || rgns.length != 0;

}

static bool lessRegionStart(IVsHiddenRegion a, IVsHiddenRegion b)

{

TextSpan aspan, bspan;

a.GetSpan(&aspan);

b.GetSpan(&bspan);

return aspan.iStartLine < bspan.iStartLine ||

(aspan.iStartLine == bspan.iStartLine && aspan.iStartIndex < bspan.iStartIndex);

}

HRESULT CollapseDisabled(bool unittests, bool disabled)

{

auto session = GetHiddenTextSession();

if(!session)

return S\_OK;

IVsEnumHiddenRegions penum;

TextSpan span = TextSpan(0, 0, 0, GetLineCount());

session.EnumHiddenRegions(FHR\_BY\_CLIENT\_DATA, kHiddenRegionCookie, &span, &penum);

mColorizer.syncParser(span.iEndLine);

IVsHiddenRegion[] rgns;

IVsHiddenRegion region;

uint fetched;

while (penum.Next(1, &region, &fetched) == S\_OK && fetched == 1)

rgns ~= region;

// sort regions by start

auto sortedrgns = sort!lessRegionStart(rgns);

int nextLine = 0;

foreach(rgn; sortedrgns)

{

DWORD state;

rgn.GetState(&state);

if((state & hrsExpanded) != 0)

{

rgn.GetSpan(&span);

int len;

if(mBuffer.GetLengthOfLine(span.iStartLine, &len) == S\_OK && span.iStartIndex >= len)

{

span.iStartLine++;

span.iStartIndex = 0;

}

if(span.iStartLine >= nextLine)

{

bool collapse = unittests && mColorizer.isInUnittest(span.iStartLine, span.iStartIndex);

if (!collapse)

collapse = disabled && !mColorizer.isAddressEnabled(span.iStartLine, span.iStartIndex);

if(collapse)

{

rgn.SetState(hrsDefault, chrDefault);

nextLine = span.iEndLine; // do not collapse recursively

}

}

}

}

foreach(rgn; rgns)

release(rgn);

release(penum);

return S\_OK;

}

///////////////////////////////////////////////////////////////////////////////

wstring GetText(int startLine, int startCol, int endLine, int endCol)

{

if(endLine == -1)

mBuffer.GetLastLineIndex(&endLine, &endCol);

else if(endCol == -1)

mBuffer.GetLengthOfLine(endLine, &endCol);

BSTR text;

HRESULT hr = mBuffer.GetLineText(startLine, startCol, endLine, endCol, &text);

return wdetachBSTR(text);

}

wstring GetText()

{

int endLine, endCol;

mBuffer.GetLastLineIndex(&endLine, &endCol);

BSTR text;

HRESULT hr = mBuffer.GetLineText(0, 0, endLine, endCol, &text);

return wdetachBSTR(text);

}

bool GetWordExtent(int line, int idx, WORDEXTFLAGS flags, out int startIdx, out int endIdx)

{

startIdx = endIdx = idx;

version(all)

{

wstring txt = GetText(line, 0, line, -1);

if(idx > txt.length)

return false;

for(size\_t p = endIdx; p < txt.length && dLex.isIdentifierCharOrDigit(decode(txt, p)); endIdx = p) {}

for(size\_t p = startIdx; p > 0 && dLex.isIdentifierCharOrDigit(decodeBwd(txt, p)); startIdx = p) {}

return startIdx < endIdx;

}

else

{

int length;

mBuffer.GetLengthOfLine(line, &length);

// pin to length of line just in case we return false and skip pinning at the end of this method.

startIdx = endIdx = min(idx, length);

if (length == 0)

return false;

//get the character classes

TokenInfo[] lineInfo = GetLineInfo(line);

if (lineInfo.length == 0)

return false;

int count = lineInfo.length;

TokenInfo info;

int index = this.GetTokenInfoAt(lineInfo, idx, info, true);

if (index < 0)

return false;

if (index < lineInfo.length - 1 && info.EndIndex == idx)

if (lineInfo[index + 1].type == TokenCat.Identifier)

info = lineInfo[++index];

if (index > 0 && info.StartIndex == idx)

if (lineInfo[index - 1].type == TokenCat.Identifier)

info = lineInfo[--index];

// don't do anything in comment or text or literal space, unless we

// are doing intellisense in which case we want to match the entire value

// of quoted strings.

TokenCat type = info.type;

if ((flags != WORDEXT\_FINDTOKEN || type != TokenCat.String) &&

(type == TokenCat.Comment || type == TokenCat.Text ||

type == TokenCat.String || type == TokenCat.Literal || type == TokenCat.Operator))

return false;

//search for a token

switch (flags & WORDEXT\_MOVETYPE\_MASK)

{

case WORDEXT\_PREVIOUS:

index--;

while (index >= 0 && !MatchToken(flags, lineInfo[index]))

index--;

if (index < 0)

return false;

break;

case WORDEXT\_NEXT:

index++;

while (index < count && !MatchToken(flags, lineInfo[index]))

index++;

if (index >= count)

return false;

break;

case WORDEXT\_NEAREST:

int prevIdx = index;

prevIdx--;

while (prevIdx >= 0 && !MatchToken(flags, lineInfo[prevIdx]))

prevIdx--;

int nextIdx = index;

while (nextIdx < count && !MatchToken(flags, lineInfo[nextIdx]))

nextIdx++;

if (prevIdx < 0 && nextIdx >= count)

return false;

if (nextIdx >= count)

index = prevIdx;

else if (prevIdx < 0)

index = nextIdx;

else if (index - prevIdx < nextIdx - index)

index = prevIdx;

else

index = nextIdx;

break;

case WORDEXT\_CURRENT:

default:

if (!MatchToken(flags, info))

return false;

break;

}

info = lineInfo[index];

// We found something, set the span, pinned to the valid coordinates for the

// current line.

startIdx = min(length, info.StartIndex);

endIdx = min(length, info.EndIndex);

return true;

}

}

bool GetTipSpan(TextSpan\* pSpan)

{

if(pSpan.iStartLine == pSpan.iEndLine && pSpan.iStartIndex == pSpan.iEndIndex)

{

int startIdx, endIdx;

if(!GetWordExtent(pSpan.iStartLine, pSpan.iStartIndex, WORDEXT\_CURRENT, startIdx, endIdx))

return false;

pSpan.iStartIndex = startIdx;

pSpan.iEndIndex = endIdx;

wstring txt = GetText(pSpan.iStartLine, 0, pSpan.iStartLine, -1);

L\_again:

size\_t idx = pSpan.iStartIndex;

dchar c;

for (size\_t p = idx; p > 0 && isWhite(c = decodeBwd(txt, p)); idx = p) {}

if(idx >= 0 && c == '.')

{

idx--; // skip '.'

for (size\_t p = idx; p > 0 && isWhite(decodeBwd(txt, p)); idx = p) {}

for (size\_t p = idx; p > 0 && dLex.isIdentifierCharOrDigit(decodeBwd(txt, p)); idx = p) {}

pSpan.iStartIndex = idx;

goto L\_again;

}

}

return true;

}

static bool MatchToken(WORDEXTFLAGS flags, TokenInfo info)

{

TokenCat type = info.type;

if ((flags & WORDEXT\_FINDTOKEN) != 0)

return type != TokenCat.Comment && type != TokenCat.String;

return (type == TokenCat.Keyword || type == TokenCat.Identifier || type == TokenCat.Literal);

}

int GetLineCount()

{

int lineCount;

mBuffer.GetLineCount(&lineCount);

return lineCount;

}

int GetLastLineIndex(ref int endLine, ref int endCol)

{

return mBuffer.GetLastLineIndex(&endLine, &endCol);

}

TokenInfo[] GetLineInfo(int line, wstring \*ptext = null)

{

wstring text = GetText(line, 0, line, -1);

if(ptext)

\*ptext = text;

return GetLineInfoFromText(line, text);

}

TokenInfo[] GetLineInfoFromText(int line, wstring text)

{

TokenInfo[] lineInfo;

int iState = mColorizer.GetLineState(line);

if(iState == -1)

return lineInfo;

lineInfo = dLex.ScanLine(iState, text);

return lineInfo;

}

static int GetTokenInfoAt(TokenInfo[] infoArray, int col, ref TokenInfo info, bool extendLast = false)

{

int len = infoArray.length;

for (int i = 0; i < len; i++)

{

int start = infoArray[i].StartIndex;

int end = infoArray[i].EndIndex;

if (i == 0 && start > col)

return -1;

if (col >= start && col < end)

{

info = infoArray[i];

return i;

}

}

if (len > 0)

{

info = infoArray[len-1];

if(col == info.EndIndex)

return len-1;

}

return -1;

}

wstring \_getToken(ref TokenInfo[] infoArray, ref int line, ref int col,

ref TokenInfo info, int idx, bool skipComments)

{

wstring text;

if(idx < 0)

idx = infoArray.length;

for(;;)

{

text = GetText(line, 0, line, -1);

while(idx < infoArray.length)

{

if((!skipComments || infoArray[idx].type != TokenCat.Comment) &&

(infoArray[idx].type != TokenCat.Text || !isWhite(text[infoArray[idx].StartIndex])))

break;

idx++;

}

if(idx < infoArray.length)

break;

line++;

int lineCount;

mBuffer.GetLineCount(&lineCount);

if(line >= lineCount)

return "";

infoArray = GetLineInfo(line);

idx = 0;

}

info = infoArray[idx];

col = infoArray[idx].StartIndex;

return text[infoArray[idx].StartIndex .. infoArray[idx].EndIndex];

}

wstring GetToken(ref TokenInfo[] infoArray, ref int line, ref int col,

ref TokenInfo info, bool skipComments = true)

{

int idx = GetTokenInfoAt(infoArray, col, info);

return \_getToken(infoArray, line, col, info, idx, skipComments);

}

wstring GetNextToken(ref TokenInfo[] infoArray, ref int line, ref int col,

ref TokenInfo info, bool skipComments = true)

{

int idx = GetTokenInfoAt(infoArray, col, info);

if(idx >= 0)

idx++;

return \_getToken(infoArray, line, col, info, idx, skipComments);

}

string GetFileName()

{

if(!mBuffer)

return null;

if(IPersistFileFormat fileFormat = qi\_cast!IPersistFileFormat(mBuffer))

{

scope(exit) release(fileFormat);

uint format;

LPOLESTR filename;

if(fileFormat.GetCurFile(&filename, &format) == S\_OK)

return to\_string(filename);

}

if(IVsUserData ud = qi\_cast!IVsUserData(mBuffer))

{

scope(exit) release(ud);

//object oname;

//Guid GUID\_VsBufferMoniker = typeof(IVsUserData).GUID;

//hr = ud.GetData(ref GUID\_VsBufferMoniker, out oname);

}

return null;

}

//////////////////////////////////////////////////////////////

bool findStatementStart(ref int line, ref int col, ref wstring fn)

{

int cl = col;

int level = 0;

TokenInfo info;

bool testNextFn = false;

for(int ln = line; ln >= 0; --ln)

{

wstring txt;

TokenInfo[] lineInfo = GetLineInfo(ln, &txt);

int inf = cl < 0 ? lineInfo.length - 1 : GetTokenInfoAt(lineInfo, cl-1, info);

for( ; inf >= 0; inf--)

{

if(lineInfo[inf].type != TokenCat.Comment &&

(lineInfo[inf].type != TokenCat.Text || !isWhite(txt[lineInfo[inf].StartIndex])))

{

wchar ch = txt[lineInfo[inf].StartIndex];

if(level == 0)

if(ch == ';' || ch == '}' || ch == '{' || ch == ':')

return true;

if(testNextFn && lineInfo[inf].type == TokenCat.Identifier)

fn = txt[lineInfo[inf].StartIndex .. lineInfo[inf].EndIndex];

testNextFn = false;

if(Lexer.isClosingBracket(ch))

level++;

else if(Lexer.isOpeningBracket(ch) && level > 0)

{

level--;

if(level == 0 && fn.length == 0)

testNextFn = true;

}

line = ln;

col = inf;

}

}

cl = -1;

}

return false;

}

wstring getScopeIdentifer(int line, int col, wstring fn)

{

TokenInfo info;

TokenInfo[] infoArray = GetLineInfo(line);

wstring next, tok = GetToken(infoArray, line, col, info);

for(;;)

{

switch(tok)

{

case "struct":

case "class":

case "interface":

case "union":

case "enum":

next = GetNextToken(infoArray, line, col, info);

if(next == ":" || next == "{")

return tok; // unnamed class/struct/enum

return next;

case "mixin":

case "static":

case "final":

case "const":

case "alias":

case "override":

case "abstract":

case "volatile":

case "deprecated":

case "in":

case "out":

case "inout":

case "lazy":

case "auto":

case "private":

case "package":

case "protected":

case "public":

case "export":

break;

case "align":

case "extern":

next = GetNextToken(infoArray, line, col, info);

if(next == "("w)

{

next = GetNextToken(infoArray, line, col, info);

next = GetNextToken(infoArray, line, col, info);

}

else

{

tok = next;

continue;

}

break;

case "synchronized":

next = GetNextToken(infoArray, line, col, info);

if(next == "("w)

{

next = GetNextToken(infoArray, line, col, info);

next = GetNextToken(infoArray, line, col, info);

}

return tok;

case "scope":

next = GetNextToken(infoArray, line, col, info);

if(next == "("w)

{

tok ~= next;

tok ~= GetNextToken(infoArray, line, col, info);

tok ~= GetNextToken(infoArray, line, col, info);

return tok;

}

break;

case "debug":

case "version":

next = GetNextToken(infoArray, line, col, info);

if(next == "("w)

{

tok ~= next;

tok ~= GetNextToken(infoArray, line, col, info);

tok ~= GetNextToken(infoArray, line, col, info);

}

return tok;

case "this":

case "if":

case "else":

case "while":

case "for":

case "do":

case "switch":

case "try":

case "catch":

case "finally":

case "with":

case "asm":

case "foreach":

case "foreach\_reverse":

return tok;

default:

return fn.length ? fn ~ "()"w : tok;

}

tok = GetNextToken(infoArray, line, col, info);

}

}

//////////////////////////////////////////////////////////////

int ReplaceLineIndent(int line, LANGPREFERENCES\* langPrefs, ref CacheLineIndentInfo cacheInfo)

{

wstring linetxt = GetText(line, 0, line, -1);

int p, orgn = countVisualSpaces(linetxt, langPrefs.uTabSize, &p);

int n = 0;

if(p < linetxt.length)

n = CalcLineIndent(line, 0, langPrefs, cacheInfo);

if(n < 0)

n = 0;

if(n == orgn)

return S\_OK;

return doReplaceLineIndent(line, p, n, langPrefs);

}

int doReplaceLineIndent(int line, int idx, int n, LANGPREFERENCES\* langPrefs)

{

int tabsz = (langPrefs.fInsertTabs && langPrefs.uTabSize > 0 ? langPrefs.uTabSize : n + 1);

string spc = replicate("\t", n / tabsz) ~ replicate(" ", n % tabsz);

wstring wspc = toUTF16(spc);

TextSpan changedSpan;

return mBuffer.ReplaceLines(line, 0, line, idx, wspc.ptr, wspc.length, &changedSpan);

}

static struct \_LineTokenIterator(SRC)

{

int line;

int tok;

SRC src;

wstring lineText;

TokenInfo[] lineInfo;

this(SRC \_src, int \_line, int \_tok)

{

src = \_src;

set(\_line, \_tok);

}

void set(int \_line, int \_tok)

{

line = \_line;

tok = \_tok;

lineInfo = src.GetLineInfo(line, &lineText);

}

bool advance()

{

while(tok + 1 >= lineInfo.length)

{

if(line + 1 >= src.GetLineCount())

return false;

line++;

lineInfo = src.GetLineInfo(line, &lineText);

tok = -1;

}

tok++;

return true;

}

bool onSpace()

{

return (lineInfo[tok].type == TokenCat.Text && isWhite(lineText[lineInfo[tok].StartIndex]));

}

bool onCommentOrSpace()

{

return (lineInfo[tok].type == TokenCat.Comment ||

(lineInfo[tok].type == TokenCat.Text && isWhite(lineText[lineInfo[tok].StartIndex])));

}

bool advanceOverSpaces()

{

while(advance())

{

if(!onSpace())

return true;

}

return false;

}

bool advanceOverComments()

{

while(advance())

{

if(!onCommentOrSpace())

return true;

}

return false;

}

bool advanceOverBraces()

{

wstring txt = getText();

if(txt == "}")

{

int otherLine, otherIndex;

if(src.FindClosingBracketForward(line, lineInfo[tok].StartIndex, otherLine, otherIndex))

{

set(otherLine, otherIndex);

}

}

return advanceOverComments();

}

bool ensureNoComment(bool skipLines)

{

if(tok < lineInfo.length && !onCommentOrSpace())

return true;

if(!skipLines)

{

while(tok + 1 < lineInfo.length)

{

tok++;

if(!onCommentOrSpace())

return true;

}

return false;

}

return advanceOverComments();

}

bool retreat()

{

while(tok <= 0)

{

if(line <= 0)

return false;

line--;

lineInfo = src.GetLineInfo(line, &lineText);

tok = lineInfo.length;

}

tok--;

return true;

}

bool retreatOverComments()

{

while(retreat())

{

if(!onCommentOrSpace())

return true;

}

return false;

}

bool retreatOverBraces()

{

wstring txt = getText();

if(txt == "}" || txt == ")" || txt == "]")

{

int otherLine, otherLinePos;

if(src.FindOpeningBracketBackward(line, tok, otherLine, otherLinePos))

{

int iState;

uint pos;

int otherIndex = src.FindLineToken(otherLine, otherLinePos, iState, pos);

set(otherLine, otherIndex);

}

}

return retreatOverComments();

}

wstring getText()

{

if(tok < lineInfo.length)

return lineText[lineInfo[tok].StartIndex .. lineInfo[tok].EndIndex];

return null;

}

int getIndex()

{

if(tok < lineInfo.length)

return lineInfo[tok].StartIndex;

return 0;

}

int getEndIndex()

{

if(tok < lineInfo.length)

return lineInfo[tok].EndIndex;

return 0;

}

int getTokenType()

{

if(tok < lineInfo.length)

return lineInfo[tok].type;

return -1;

}

int getTokenId()

{

if(tok < lineInfo.length)

return lineInfo[tok].tokid;

return -1;

}

wstring getPrevToken(int n = 1)

{

auto it = this;

foreach(i; 0..n)

it.retreatOverComments();

return it.getText();

}

wstring getNextToken(int n = 1)

{

auto it = this;

foreach(i; 0..n)

it.advanceOverComments();

return it.getText();

}

}

alias \_LineTokenIterator!Source LineTokenIterator;

static struct CacheLineIndentInfo

{

bool hasOpenBraceInfoValid;

bool hasOpenBrace;

int hasOpenBraceLine;

int hasOpenBraceTok;

LineTokenIterator hasOpenBraceIt;

bool findCommaInfoValid;

int findCommaIndent;

int findCommaIndentLine;

}

// calculate the indentation of the given line

// - if ch != 0, assume it being inserted at the beginning of the line

// - find the beginning of the previous statement

// - if the first token on the line is "else", find the matching "if" and indent to its line

// - set iterator tokIt to the last token of the previous line

// - if \*tokIt is ';', move back one

// - while \*tokIt is not the stop marker or '{' or ';'

// - move back one matching braces

// - if the token before the given line is not ';' or '}', indent by one level more

// special handling for:

// - comma at the end of next line

// - case/default

// - label:

int CalcLineIndent(int line, dchar ch, LANGPREFERENCES\* langPrefs, ref CacheLineIndentInfo cacheInfo)

{

LineTokenIterator lntokIt = LineTokenIterator(this, line, 0);

wstring startTok;

if(ch != 0)

startTok ~= ch;

else

{

lntokIt.ensureNoComment(false);

startTok = lntokIt.getText();

}

wstring txt;

if(!lntokIt.retreatOverComments())

return 0;

bool isOpenBraceOrCase(ref LineTokenIterator it)

{

wstring txt = it.getText();

if(txt == "{" || txt == "[")

return true;

if(txt == "case" || txt == "default")

{

wstring prev = it.getPrevToken();

if(prev != "goto")

return true;

}

return false;

}

int findMatchingIf()

{

int cntIf = 1;

while(cntIf > 0 && lntokIt.retreatOverBraces())

{

if(isOpenBraceOrCase(lntokIt)) // emergency exit on pending opening brace

return countVisualSpaces(lntokIt.lineText, langPrefs.uTabSize) + langPrefs.uTabSize;

txt = lntokIt.getText();

if(txt == "if")

--cntIf;

else if(txt == "else")

++cntIf;

}

return countVisualSpaces(lntokIt.lineText, langPrefs.uTabSize);

}

bool findOpenBrace(ref LineTokenIterator it)

{

int itline = it.line;

int ittok = it.tok;

bool saveCacheInfo(bool res)

{

cacheInfo.hasOpenBraceInfoValid = true;

cacheInfo.hasOpenBrace = res;

cacheInfo.hasOpenBraceIt = it;

cacheInfo.hasOpenBraceLine = itline;

cacheInfo.hasOpenBraceTok = ittok;

return res;

}

do

{

txt = it.getText();

if(txt == "{" || txt == "[" || txt == "(")

return saveCacheInfo(true);

if(cacheInfo.hasOpenBraceInfoValid && it.line == cacheInfo.hasOpenBraceLine && it.tok == cacheInfo.hasOpenBraceTok)

{

it = cacheInfo.hasOpenBraceIt;

return cacheInfo.hasOpenBrace;

}

}

while(it.retreatOverBraces());

return saveCacheInfo(false);

}

int findPreviousCaseIndent()

{

do

{

txt = lntokIt.getText();

if(txt == "{" || txt == "[") // emergency exit on pending opening brace

return countVisualSpaces(lntokIt.lineText, langPrefs.uTabSize) + langPrefs.uTabSize;

if(txt == "case" || txt == "default") // emergency exit on pending opening brace

if(lntokIt.getPrevToken() != "goto")

break;

}

while(lntokIt.retreatOverBraces());

return countVisualSpaces(lntokIt.lineText, langPrefs.uTabSize);

}

// called when previous line ends with a comma

// use cases:

//

// enum ID {

// E1,

//--------------

// function(arg1,

//--------------

// int[] arr = [

// expression,

//--------------

// Struct s = {

// expression,

//--------------

// case C1,

//--------------

// case C1:

// expression,

//--------------

// label:

// expression,

//--------------

// public import mod1,

//--------------

// ulong var,

//--------------

// const(UDT) var,

int findCommaIndent()

{

int itline = lntokIt.line;

int saveCacheInfo(int indent)

{

cacheInfo.findCommaInfoValid = true;

cacheInfo.findCommaIndent = indent;

cacheInfo.findCommaIndentLine = itline;

return indent;

}

wstring txt;

int commaIndent = countVisualSpaces(lntokIt.lineText, langPrefs.uTabSize);

do

{

if(cacheInfo.findCommaInfoValid && lntokIt.line < cacheInfo.findCommaIndentLine)

return saveCacheInfo(cacheInfo.findCommaIndent);

txt = lntokIt.getText();

if(txt == "(")

// TODO: should scan for first non-white after '('

return saveCacheInfo(visiblePosition(lntokIt.lineText, langPrefs.uTabSize, lntokIt.getIndex() + 1));

if(txt == "[")

return saveCacheInfo(commaIndent);

if(txt == ",")

commaIndent = countVisualSpaces(lntokIt.lineText, langPrefs.uTabSize);

if(txt == "{")

{

// figure out if this is a struct initializer, enum declaration or a statement group

if(lntokIt.retreatOverBraces())

{

wstring prev = txt;

txt = lntokIt.getText();

if(txt == "=") // struct initializer

return saveCacheInfo(commaIndent);

do

{

txt = lntokIt.getText();

if(txt == "{" || txt == "}" || txt == ";")

{

if(prev == "enum")

return saveCacheInfo(commaIndent);

else

break;

}

prev = txt;

}

while(lntokIt.retreatOverBraces());

}

return saveCacheInfo(commaIndent + langPrefs.uTabSize);

}

if(isOpenBraceOrCase(lntokIt))

return saveCacheInfo(countVisualSpaces(lntokIt.lineText, langPrefs.uTabSize) + langPrefs.uTabSize);

if(txt == "}" || txt == ";") // triggers the end of a statement, but not do {} while()

{

// indent once from line with first comma

return saveCacheInfo(commaIndent + langPrefs.uTabSize);

//                                        lntokIt.advanceOverComments();

//                                        return countVisualSpaces(lntokIt.lineText, langPrefs.uTabSize) + langPrefs.uTabSize;

}

}

while(lntokIt.retreatOverBraces());

return saveCacheInfo(0);

}

if(startTok == "else")

return findMatchingIf();

if(startTok == "case" || startTok == "default")

return findPreviousCaseIndent();

LineTokenIterator it = lntokIt;

bool hasOpenBrace = findOpenBrace(it);

if(hasOpenBrace && txt == "(")

{

LineTokenIterator nit = it;

if(nit.advanceOverSpaces() && nit.line < line)

return visiblePosition(nit.lineText, langPrefs.uTabSize, nit.getIndex());

}

if(startTok == "}" || startTok == "]")

{

if(hasOpenBrace)

return countVisualSpaces(it.lineText, langPrefs.uTabSize);

return 0;

}

wstring prevTok = lntokIt.getText();

if(prevTok == ",")

return findCommaIndent();

int indent = 0, labelIndent = 0;

bool newStmt = (prevTok == ";" || prevTok == "}" || prevTok == "{" || prevTok == ":");

if(newStmt)// || prevTok == ":")

if(dLex.isIdentifier(startTok) && lntokIt.getNextToken(2) == ":") // is it a jump label?

{

labelIndent = -langPrefs.uTabSize;

newStmt = true;

}

if(newStmt)

{

if(prevTok != "{" && prevTok != ":")

lntokIt.retreatOverBraces();

}

else if(prevTok == ")" && (startTok == "in" || startTok == "out" || startTok == "body"))

indent = 0; // special case to not indent in/out/body contracts

else if(startTok != "{" && startTok != "[" && hasOpenBrace)

indent = langPrefs.uTabSize;

if(prevTok == "{" || prevTok == "[")

return countVisualSpaces(lntokIt.lineText, langPrefs.uTabSize) + langPrefs.uTabSize + labelIndent;

bool skipLabel = false;

do

{

txt = lntokIt.getText();

if(txt == "(")

return visiblePosition(lntokIt.lineText, langPrefs.uTabSize, lntokIt.getIndex() + 1);

if(isOpenBraceOrCase(lntokIt))

return countVisualSpaces(lntokIt.lineText, langPrefs.uTabSize) + langPrefs.uTabSize + indent + labelIndent;

if(txt == "}" || txt == ";") // triggers the end of a statement, but not do {} while()

{

// use indentation of next statement

lntokIt.advanceOverComments();

// skip labels

wstring label = lntokIt.getText();

if(!dLex.isIdentifier(label) || lntokIt.getNextToken() != ":")

return countVisualSpaces(lntokIt.lineText, langPrefs.uTabSize) + indent + labelIndent;

lntokIt.retreatOverComments();

newStmt = true;

}

if(!newStmt && isKeyword(toUTF8(txt))) // dLex.isIdentifier(txt))

{

return indent + countVisualSpaces(lntokIt.lineText, langPrefs.uTabSize);

}

if(newStmt && txt == "else")

{

findMatchingIf();

if(isOpenBraceOrCase(lntokIt))

return countVisualSpaces(lntokIt.lineText, langPrefs.uTabSize) + langPrefs.uTabSize + labelIndent;

}

}

while(lntokIt.retreatOverBraces());

return indent + labelIndent;

}

int ReindentLines(IVsTextView view, int startline, int endline)

{

LANGPREFERENCES langPrefs;

if(int rc = GetUserPreferences(&langPrefs, view))

return rc;

if(langPrefs.IndentStyle != vsIndentStyleSmart)

return S\_FALSE;

CacheLineIndentInfo cacheInfo;

for(int line = startline; line <= endline; line++)

{

int rc = ReplaceLineIndent(line, &langPrefs, cacheInfo);

if(FAILED(rc))

return rc;

}

return S\_OK;

}

////////////////////////////////////////////////////////////////////////

wstring FindExpressionBefore(int caretLine, int caretIndex)

{

int startLine, startIndex;

LineTokenIterator lntokIt = LineTokenIterator(this, caretLine + 1, 0);

while(lntokIt.line > caretLine || (lntokIt.getIndex() >= caretIndex && lntokIt.line == caretLine))

if(!lntokIt.retreatOverComments())

break;

if(lntokIt.getTokenType() == TokenColor.Identifier && lntokIt.getEndIndex() >= caretIndex && lntokIt.line == caretLine)

lntokIt.retreatOverComments();

if(lntokIt.getText() != ".")

return null;

caretLine = lntokIt.line;

caretIndex = lntokIt.getIndex();

lntokIt.retreatOverComments();

L\_retry:

startLine = lntokIt.line;

startIndex = lntokIt.getIndex();

int type = lntokIt.getTokenType();

if(type == TokenColor.Identifier || type == TokenColor.String || type == TokenColor.Literal)

{

lntokIt.retreatOverComments();

wstring tok = lntokIt.getText();

if(tok == "." || tok == "!")

{

lntokIt.retreatOverComments();

goto L\_retry;

}

}

else

{

wstring tok = lntokIt.getText();

if(tok == "}" || tok == ")" || tok == "]")

{

lntokIt.retreatOverBraces();

goto L\_retry;

}

}

wstring wsnip = GetText(startLine, startIndex, caretLine, caretIndex);

return wsnip;

}

////////////////////////////////////////////////////////////////////////

enum

{

AutoComment,

ForceComment,

ForceUncomment,

}

int CommentLines(IVsTextView view, int startline, int endline, int commentMode)

{

LANGPREFERENCES langPrefs;

if(int rc = GetUserPreferences(&langPrefs, view))

return rc;

wstring[] lines;

wstring txt;

int n, m, p, indent = -1;

int line;

// calc minimum indent

for(line = startline; line <= endline; line++)

{

txt = GetText(line, 0, line, -1);

n = countVisualSpaces(txt, langPrefs.uTabSize, &p);

if (p < txt.length) // ignore empty line

indent = (indent < 0 || indent > n ? n : indent);

lines ~= txt;

}

for(line = startline; line <= endline; line++)

{

txt = lines[line - startline];

n = countVisualSpaces(txt, langPrefs.uTabSize, &p);

if(p >= txt.length || n != indent)

break;

else if(p + 1 >= txt.length || txt[p] != '/' || txt[p+1] != '/')

break;

}

if (line > endline && commentMode != ForceComment)

{

// remove comment

for(line = startline; line <= endline; line++)

{

txt = lines[line - startline];

n = countVisualSpaces(txt, langPrefs.uTabSize, &p);

assert(n == indent && txt[p] == '/' && txt[p+1] == '/');

txt = txt[0..p] ~ " " ~ txt[p+2..$];

m = countVisualSpaces(txt, langPrefs.uTabSize, &p) - 2;

if(p >= txt.length)

txt = "";

else

txt = createVisualSpaces!wstring(m, langPrefs.fInsertTabs ? langPrefs.uTabSize : 0);

TextSpan changedSpan;

if (int hr = mBuffer.ReplaceLines(line, 0, line, p, txt.ptr, txt.length, &changedSpan))

return hr;

}

}

else if((line <= endline && commentMode != ForceUncomment) || commentMode == ForceComment)

{

// insert comment

int tabsz = (langPrefs.fInsertTabs ? langPrefs.uTabSize : 0);

wstring pfx = createVisualSpaces!wstring(indent, tabsz) ~ "//"w;

for(line = startline; line <= endline; line++)

{

txt = lines[line - startline];

n = countVisualSpaces(txt, langPrefs.uTabSize, &p);

wstring add = createVisualSpaces!wstring(n - indent, 0, 2); // use spaces, not tabs

wstring ins = pfx ~ add;

TextSpan changedSpan;

if (int hr = mBuffer.ReplaceLines(line, 0, line, p, ins.ptr, ins.length, &changedSpan))

return hr;

}

}

return S\_OK;

}

//////////////////////////////////////////////////////////////

// return the token index from the scan sequence

// iState,pos is the scan state before the token at char index idx

int FindLineToken(int line, int idx, out int iState, out uint pos)

{

int state = mColorizer.GetLineState(line);

if(state == -1)

return -1;

wstring text = GetText(line, 0, line, -1);

uint p = 0;

int tok = 0;

while(p < text.length)

{

iState = state;

pos = p;

if(p == idx)

return tok;

dLex.scan(state, text, p);

if(p > idx)

return tok;

tok++;

}

return -1;

}

// continuing from FindLineToken

bool FindEndOfTokens(ref int iState, ref int line, ref uint pos,

bool function(int state, int data) testFn, int data)

{

int lineCount;

mBuffer.GetLineCount(&lineCount);

uint plinepos = pos;

while(line < lineCount)

{

wstring text = GetText(line, 0, line, -1);

while(pos < text.length)

{

uint ppos = pos;

int toktype = dLex.scan(iState, text, pos);

if(testFn(iState, data))

{

/+

if(ppos == 0)

{

pos = plinepos;

line--;

}

else

pos = ppos;

+/

return true;

}

}

plinepos = pos;

pos = 0;

line++;

}

return false;

}

static bool testEndComment(int state, int level)

{

int slevel = Lexer.nestingLevel(state);

if(slevel > level)

return false;

auto sstate = Lexer.scanState(state);

if(sstate == Lexer.State.kNestedComment)

return slevel <= level;

return sstate != Lexer.State.kBlockComment;

}

bool FindEndOfComment(int startState, ref int iState, ref int line, ref uint pos)

{

int level = Lexer.nestingLevel(startState);

if(testEndComment(iState, level))

return true;

return FindEndOfTokens(iState, line, pos, &testEndComment, level);

}

static bool testEndString(int state, int level)

{

if(Lexer.tokenStringLevel(state) > level)

return false;

auto sstate = Lexer.scanState(state);

return !Lexer.isStringState(sstate);

}

bool FindEndOfString(int startState, ref int iState, ref int line, ref uint pos)

{

int level = Lexer.tokenStringLevel(startState);

if(testEndString(iState, level))

return true;

return FindEndOfTokens(iState, line, pos, &testEndString, level);

}

bool FindStartOfTokens(ref int iState, ref int line, ref uint pos,

bool function(int state, int data) testFn, int data)

{

int lineState;

uint plinepos = pos;

uint foundpos = uint.max;

while(line >= 0)

{

wstring text = GetText(line, 0, line, -1);

lineState = mColorizer.GetLineState(line);

uint len = (plinepos > text.length ? text.length : plinepos);

plinepos = 0;

if(testFn(lineState, data))

foundpos = 0;

while(plinepos < len)

{

int toktype = dLex.scan(lineState, text, plinepos);

if(testFn(lineState, data))

foundpos = plinepos;

}

if(foundpos < uint.max)

{

pos = foundpos;

return true;

}

plinepos = uint.max;

line--;

}

return false;

}

static bool testStartComment(int state, int level)

{

if(!Lexer.isCommentState(Lexer.scanState(state)))

return true;

int slevel = Lexer.nestingLevel(state);

return slevel < level;

}

bool FindStartOfComment(ref int iState, ref int line, ref uint pos)

{

// comment ends after the token that starts at (line,pos) with state iState

// possible states:

// - not a comment state: comment starts at passed pos

// - it's a block comment: scan backwards until we find a non-comment state

// - it's a nested comment: scan backwards until we find a state with nesting level less than passed state

if(!Lexer.isCommentState(Lexer.scanState(iState)))

return true;

int level = Lexer.nestingLevel(iState);

return FindStartOfTokens(iState, line, pos, &testStartComment, level);

}

bool FindStartOfString(ref int iState, ref int line, ref uint pos)

{

int level = Lexer.tokenStringLevel(iState);

if(testEndString(iState, level))

return true;

return FindStartOfTokens(iState, line, pos, &testEndString, level);

}

bool FindClosingBracketForward(int line, int idx, out int otherLine, out int otherIndex)

{

int iState;

uint pos;

int tok = FindLineToken(line, idx, iState, pos);

if(tok < 0)

return false;

wstring text = GetText(line, 0, line, -1);

uint ppos = pos;

int toktype = dLex.scan(iState, text, pos);

if(toktype != TokenCat.Operator)

return false;

return FindClosingBracketForward(line, iState, pos, otherLine, otherIndex);

}

bool FindClosingBracketForward(int line, int iState, uint pos, out int otherLine, out int otherIndex)

{

int lineCount;

mBuffer.GetLineCount(&lineCount);

int level = 1;

while(line < lineCount)

{

wstring text = GetText(line, 0, line, -1);

while(pos < text.length)

{

uint ppos = pos;

int type = dLex.scan(iState, text, pos);

if(type == TokenCat.Operator)

{

if(Lexer.isOpeningBracket(text[ppos]))

level++;

else if(Lexer.isClosingBracket(text[ppos]))

if(--level <= 0)

{

otherLine = line;

otherIndex = ppos;

return true;

}

}

}

line++;

pos = 0;

}

return false;

}

bool FindOpeningBracketBackward(int line, int tok, out int otherLine, out int otherIndex,

int\* pCountComma = null)

{

if(pCountComma)

\*pCountComma = 0;

int level = 1;

while(line >= 0)

{

wstring text = GetText(line, 0, line, -1);

int[] tokpos;

int[] toktype;

uint pos = 0;

int iState = mColorizer.GetLineState(line);

if(iState == -1)

break;

while(pos < text.length)

{

tokpos ~= pos;

toktype ~= dLex.scan(iState, text, pos);

}

int p = (tok >= 0 ? tok : tokpos.length) - 1;

for( ; p >= 0; p--)

{

pos = tokpos[p];

if(toktype[p] == TokenCat.Operator)

{

if(pCountComma && text[pos] == ',')

(\*pCountComma)++;

else if(Lexer.isClosingBracket(text[pos]))

level++;

else if(Lexer.isOpeningBracket(text[pos]))

if(--level <= 0)

{

otherLine = line;

otherIndex = pos;

return true;

}

}

}

line--;

tok = -1;

}

return false;

}

bool ScanBackward(int line, int tok,

bool delegate(wstring text, uint pos, uint ppos, int type) dg)

{

while(line >= 0)

{

wstring text = GetText(line, 0, line, -1);

int[] tokpos;

int[] toktype;

uint pos = 0;

int iState = mColorizer.GetLineState(line);

if(iState == -1)

break;

while(pos < text.length)

{

tokpos ~= pos;

toktype ~= dLex.scan(iState, text, pos);

}

int p = (tok >= 0 ? tok : tokpos.length) - 1;

uint ppos = (p >= tokpos.length - 1 ? text.length : tokpos[p+1]);

for( ; p >= 0; p--)

{

pos = tokpos[p];

if(dg(text, pos, ppos, toktype[p]))

return true;

ppos = pos;

}

line--;

tok = -1;

}

return false;

}

// tok is sitting on the opening parenthesis, return method name and its position

wstring FindMethodIdentifierBackward(int line, int tok, int\* pline, int\* pindex)

{

LineTokenIterator it = LineTokenIterator(this, line, tok);

scope(exit)

{

if(pline)

\*pline = it.line;

if(pindex)

\*pindex = it.getIndex();

}

if(!it.retreatOverComments())

return null;

if(it.getText() == ")")

{

// skip over template arguments

if(it.retreatOverBraces() &&

it.getText() == "!" &&

it.retreatOverComments() &&

it.getTokenType() == TokenCat.Identifier)

return it.getText();

return null;

}

if(it.getText() == "!")

{

// inside template argument list

if(it.retreatOverComments() &&

it.getTokenType() == TokenCat.Identifier)

return it.getText();

return null;

}

switch(it.getTokenId())

{

case TOK\_\_\_vector:

mixin(case\_TOKs\_BasicTypeX);

mixin(case\_TOKs\_TemplateSingleArgument);

{

LineTokenIterator it2 = it;

if(it2.retreatOverComments() &&

it2.getText() == "!" &&

it2.retreatOverComments())

it = it2;

break;

}

default:

break;

}

if (it.getTokenType() == TokenCat.Identifier)

return it.getText();

return null;

}

//////////////////////////////////////////////////////////////

class ClippingSource

{

Source mSrc;

int mClipLine;

int mClipIndex;

this(Source src)

{

mSrc = src;

mClipLine = int.max;

}

void setClip(int line, int idx)

{

mClipLine = line;

mClipIndex = idx;

}

int GetLineCount()

{

int lines = mSrc.GetLineCount();

if(lines - 1 > mClipLine)

lines = mClipLine + 1;

return lines;

}

TokenInfo[] GetLineInfo(int line, wstring \*ptext = null)

{

if(line > mClipLine)

return null;

if(line < mClipLine)

return mSrc.GetLineInfo(line, ptext);

wstring text = GetText(line, 0, line, -1);

if(text.length > mClipIndex)

text = text[0 .. mClipIndex];

if(ptext)

\*ptext = text;

return mSrc.GetLineInfoFromText(line, text);

}

int FindLineToken(int line, int idx, out int iState, out uint pos)

{

// only used in brace matched search

return mSrc.FindLineToken(line, idx, iState, pos);

}

bool FindOpeningBracketBackward(int line, int tok, out int otherLine, out int otherIndex,

int\* pCountComma = null)

{

// no brace matching needed for finding imports

return mSrc.FindOpeningBracketBackward(line, tok, otherLine, otherIndex, pCountComma);

}

bool FindClosingBracketForward(int line, int idx, out int otherLine, out int otherIndex)

{

// no brace matching needed for finding imports

return mSrc.FindClosingBracketForward(line, idx, otherLine, otherIndex);

}

}

wstring GetImportModule(int line, int index, bool clipSource)

{

auto clipsrc = new ClippingSource(this);

if(clipSource)

clipsrc.setClip(line, index);

auto lntokIt = \_LineTokenIterator!ClippingSource(clipsrc, line, 0);

while(lntokIt.line < line || (lntokIt.getIndex() <= index && lntokIt.line == line))

if (!lntokIt.advanceOverComments())

goto L\_eol;

lntokIt.retreatOverComments();

L\_eol:

wstring tok = lntokIt.getText();

while((tok == "static" || tok == "public" || tok == "private")

&& lntokIt.advanceOverComments())

tok = lntokIt.getText();

while(tok != "import" && (tok == "." || dLex.isIdentifier(tok))

&& lntokIt.retreatOverComments())

tok = lntokIt.getText();

auto lntokIt2 = lntokIt;

while(tok != "import" && (tok == "," || tok == "=" || tok == ":" || tok == "." || dLex.isIdentifier(tok))

&& lntokIt.retreatOverComments())

{

if(tok == ":")

return null; // no import handling on selective import identifier

tok = lntokIt.getText();

}

if(tok != "import")

return null;

lntokIt2.advanceOverComments();

tok = lntokIt2.getText();

wstring imp;

while(tok == "." || dLex.isIdentifier(tok))

{

imp ~= tok;

if(!lntokIt2.advanceOverComments())

break;

tok = lntokIt2.getText();

}

return imp;

}

//////////////////////////////////////////////////////////////

// create our own task pool to be able to destroy it (it keeps a the

// arguments to the last task, so they are never collected)

\_\_gshared TaskPool parseTaskPool;

void runTask(T)(T dg)

{

if(!parseTaskPool)

{

int threads = defaultPoolThreads;

if(threads < 1)

threads = 1;

parseTaskPool = new TaskPool(threads);

parseTaskPool.isDaemon = true;

parseTaskPool.priority(core.thread.Thread.PRIORITY\_MIN);

}

auto task = task(dg);

parseTaskPool.put(task);

}

bool startParsing()

{

if(!Package.GetGlobalOptions().parseSource && !mOutlining)

return false;

if(mParsingState > 1)

return finishParsing();

if(mParsingState != 0 || mModificationCountAST == mModificationCount)

return false;

bool verbose = (mModificationCountAST == -1);

mParseText = GetText(); // should not be read from another thread

mParsingState = 1;

mModificationCountAST = mModificationCount;

runTask(&doParse);

if(Package.GetGlobalOptions().parseSource)

{

auto langsvc = Package.GetLanguageService();

langsvc.vdServerClient.UpdateModule(GetFileName(), mParseText, verbose, &OnUpdateModule);

}

return true;

}

bool ensureCurrentTextParsed()

{

if(mModificationCountAST != mModificationCount)

return startParsing();

return false;

}

extern(D) void OnUpdateModule(uint request, string filename, string parseErrors, vdc.util.TextPos[] binaryIsIn)

{

updateParseErrors(parseErrors);

mBinaryIsIn = binaryIsIn;

if(IVsTextColorState colorState = qi\_cast!IVsTextColorState(mBuffer))

{

scope(exit) release(colorState);

foreach(pos; mBinaryIsIn)

colorState.ReColorizeLines(pos.line - 1, pos.line - 1);

}

}

void updateParseErrors(string err)

{

string[] errs = splitLines(err);

mParseErrors = mParseErrors.init;

foreach(e; errs)

{

auto idx = indexOf(e, ':');

if(idx > 0)

{

string[] num = split(e[0..idx], ",");

if(num.length == 4)

{

try

{

ParseError error;

error.span.iStartLine = parse!int(num[0]);

error.span.iStartIndex = parse!int(num[1]);

error.span.iEndLine = parse!int(num[2]);

error.span.iEndIndex = parse!int(num[3]);

error.msg = e[idx+1..$];

mParseErrors ~= error;

}

catch(ConvException)

{

}

}

}

}

finishParseErrros();

}

void finishParseErrros()

{

IVsEnumLineMarkers pEnum;

if(mBuffer.EnumMarkers(0, 0, 0, 0, MARKER\_CODESENSE\_ERROR, EM\_ENTIREBUFFER, &pEnum) == S\_OK)

{

scope(exit) release(pEnum);

IVsTextLineMarker marker;

while(pEnum.Next(&marker) == S\_OK)

{

marker.Invalidate();

marker.Release();

}

}

for(int i = 0; i < mParseErrors.length; i++)

{

auto span = mParseErrors[i].span;

IVsTextLineMarker marker;

mBuffer.CreateLineMarker(MARKER\_CODESENSE\_ERROR, span.iStartLine - 1, span.iStartIndex,

span.iEndLine - 1, span.iEndIndex, this, &marker);

}

}

bool finishParsing()

{

if(mOutlining)

{

if(mStopOutlining)

{

mOutlineRegions = mOutlineRegions.init;

mOutlining = false;

}

if(mModificationCountAST == mModificationCount)

{

if(auto session = GetHiddenTextSession())

if(DiffRegions(session, mOutlineRegions))

session.AddHiddenRegions(chrNonUndoable, mOutlineRegions.length, mOutlineRegions.ptr, null);

mOutlineRegions = mOutlineRegions.init;

}

}

mParseText = null;

mParsingState = 0;

ReColorizeLines(0, -1);

return true;

}

void doParse()

{

if(mOutlining)

{

mOutlineRegions = CreateOutlineRegions(mParseText, hrsExpanded);

}

mParsingState = 2;

}

bool hasParseError(ParserSpan span)

{

for(int i = 0; i < mParseErrors.length; i++)

if(spanContains(span, mParseErrors[i].span.iStartLine-1, mParseErrors[i].span.iStartIndex))

return true;

return false;

}

string getParseError(int line, int index)

{

for(int i = 0; i < mParseErrors.length; i++)

if(spanContains(mParseErrors[i].span, line+1, index))

return mParseErrors[i].msg;

return null;

}

//////////////////////////////////////////////////////////////

ExpansionProvider GetExpansionProvider()

{

if(!mExpansionProvider)

mExpansionProvider = addref(newCom!ExpansionProvider(this));

return mExpansionProvider;

}

IVsTextLines GetTextLines() { return mBuffer; }

CompletionSet GetCompletionSet()

{

if(!mCompletionSet)

mCompletionSet = addref(newCom!CompletionSet(null, this));

return mCompletionSet;

}

MethodData GetMethodData()

{

if(!mMethodData)

mMethodData = addref(newCom!MethodData());

return mMethodData;

}

bool IsCompletorActive()

{

if (mCompletionSet && mCompletionSet.mDisplayed)

return true;

return false;

}

bool IsMethodTipActive()

{

if (mMethodData && mMethodData.mDisplayed)

return true;

return false;

}

void DismissCompletor()

{

if (mCompletionSet && mCompletionSet.mDisplayed)

mCompletionSet.Close();

}

void DismissMethodTip()

{

if (mMethodData && mMethodData.mDisplayed)

mMethodData.Close();

}

bool EnableFormatSelection() { return true; }

}

///////////////////////////////////////////////////////////////////////////////

class EnumProximityExpressions : DComObject, IVsEnumBSTR

{

wstring[] mExpressions;

int mPos;

this(Source src, int iLine, int iCol, int cLines)

{

int begLine = iLine < cLines ? 0 : iLine - cLines;

for(int line = begLine; line < iLine + cLines; line++)

{

int iState = src.mColorizer.GetLineState(line);

if(iState == -1)

break;

wstring text = src.GetText(line, 0, line, -1);

uint pos = 0;

wstring ident;

while(pos < text.length)

{

uint ppos = pos;

int type = dLex.scan(iState, text, pos);

wstring txt = text[ppos .. pos];

if(type == TokenCat.Identifier || txt == "this"w)

{

ident ~= txt;

if(ident.length > 4 && ident[0..5] == "this."w)

ident = "this->"w ~ ident[5..$];

addunique(mExpressions, ident);

//                                        if(!ident.startsWith("this."w))

//                                                addunique(mExpressions, "this."w ~ ident);

}

else if (type == TokenCat.Operator && txt == "."w)

ident ~= "."w;

else

ident = ""w;

}

}

if(arrIndex(mExpressions, "this"w) < 0)

mExpressions ~= "this"w;

}

this(EnumProximityExpressions epe)

{

mExpressions = epe.mExpressions;

mPos = epe.mPos;

}

override HRESULT QueryInterface(in IID\* riid, void\*\* pvObject)

{

if(queryInterface!(IVsEnumBSTR) (this, riid, pvObject))

return S\_OK;

return super.QueryInterface(riid, pvObject);

}

// IVsEnumBSTR

override int Next(in ULONG celt, BSTR \*rgelt, ULONG \*pceltFetched)

{

if(mPos + celt > mExpressions.length)

return E\_FAIL;

for(int i = 0; i < celt; i++)

rgelt[i] = allocwBSTR(mExpressions[mPos + i]);

mPos += celt;

if(pceltFetched)

\*pceltFetched = celt;

return S\_OK;

}

override int Skip(in ULONG celt)

{

mPos += celt;

return S\_OK;

}

override int Reset()

{

mPos = 0;

return S\_OK;

}

override int Clone(IVsEnumBSTR\* ppenum)

{

auto clone = newCom!EnumProximityExpressions(this);

\*ppenum = addref(clone);

return S\_OK;

}

override int GetCount(ULONG \*pceltCount)

{

\*pceltCount = mExpressions.length;

return S\_OK;

}

}

///////////////////////////////////////////////////////////////////////

// This file is part of Visual D

//

// Visual D integrates the D programming language into Visual Studio

// Copyright (c) 2010 by Rainer Schuetze, All Rights Reserved

//

// Distributed under the Boost Software License, Version 1.0.

// See accompanying file LICENSE\_1\_0.txt or copy at <http://www.boost.org/LICENSE_1_0.txt>

module visuald.dllmain;

import stdwin = core.sys.windows.windows;

import visuald.windows;

import visuald.comutil;

import visuald.logutil;

import visuald.register;

import visuald.dpackage;

import visuald.dlangsvc;

import visuald.getmsobj;

import std.parallelism;

import core.runtime;

import core.memory;

import core.sys.windows.dll;

import threadaux = core.sys.windows.threadaux;

import std.conv;

\_\_gshared HINSTANCE g\_hInst;

///////////////////////////////////////////////////////////////////////

//version = MAIN;

version(MAIN)

{

int main()

{

return VerifyMSObj(("Software\\Microsoft\\VisualStudio\\9.0D"w).ptr);

//return VSDllRegisterServer(("Software\\Microsoft\\VisualStudio\\9.0D"w).ptr);

//return VSDllUnregisterServerUser(("Software\\Microsoft\\VisualStudio\\9.0D"w).ptr);

}

}

else version(TESTMAIN)

{

import vdc.semantic;

\_\_gshared extern(C) extern long gcdump\_userData;

\_\_gshared extern(C) extern bool gcdump\_pools;

int main()

{

Project prj = new Project;

string[] imps = [ r"m:\s\d\rainers\druntime\import\", r"m:\s\d\rainers\phobos\" ];

string fname = r"m:\s\d\rainers\phobos\std\datetime.d";

prj.options.setImportDirs(imps);

prj.addAndParseFile(fname);

//                gcdump\_pools = true;

//                GC.collect();

//                gcdump\_pools = false;

//                prj.semantic();

foreach(i; 1..100)

{

//                        gcdump\_userData = i;

prj.addAndParseFile(fname);

}

return 0;

}

}

else // !version(TESTMAIN)

{

} // !version(D\_Version2)

void clearStack()

{

// fill stack with zeroes, so the chance of having false pointers is reduced

int[1000] arr;

}

version(MAIN) {} else version(TESTMAIN) {} else

extern (Windows)

BOOL DllMain(stdwin.HINSTANCE hInstance, ULONG ulReason, LPVOID pvReserved)

{

switch (ulReason)

{

case DLL\_PROCESS\_ATTACH:

//MessageBoxA(cast(HANDLE)0, "Hi", "there", 0);

if(!dll\_process\_attach(hInstance, true))

return false;

g\_hInst = cast(HINSTANCE) hInstance;

//        GC.disable();

global\_init();

logCall("DllMain(DLL\_PROCESS\_ATTACH, tid=%x)", GetCurrentThreadId());

break;

case DLL\_PROCESS\_DETACH:

logCall("DllMain(DLL\_PROCESS\_DETACH, tid=%x)", GetCurrentThreadId());

global\_exit();

debug clearStack();

debug GC.collect();

debug DComObject.showCOMleaks();

dll\_process\_detach(hInstance, true);

debug if(DComObject.sCountReferenced != 0 || DComObject.sCountInstances != 0)

asm { int 3; } // use continue, not terminate in the debugger

break;

case DLL\_THREAD\_ATTACH:

if(!dll\_thread\_attach(true, true))

return false;

logCall("DllMain(DLL\_THREAD\_ATTACH, id=%x)", GetCurrentThreadId());

break;

case DLL\_THREAD\_DETACH:

if(threadaux.GetTlsDataAddress(GetCurrentThreadId())) //, \_tls\_index))

logCall("DllMain(DLL\_THREAD\_DETACH, id=%x)", GetCurrentThreadId());

dll\_thread\_detach(true, true);

break;

default:

assert(\_false);

return false;

}

return true;

}

extern (Windows)

void RunDLLRegister(HWND hwnd, HINSTANCE hinst, LPSTR lpszCmdLine, int nCmdShow)

{

wstring ws = to\_wstring(lpszCmdLine) ~ cast(wchar)0;

VSDllRegisterServer(ws.ptr);

}

extern (Windows)

void RunDLLUnregister(HWND hwnd, HINSTANCE hinst, LPSTR lpszCmdLine, int nCmdShow)

{

wstring ws = to\_wstring(lpszCmdLine) ~ cast(wchar)0;

VSDllUnregisterServer(ws.ptr);

}

extern (Windows)

void RunDLLRegisterUser(HWND hwnd, HINSTANCE hinst, LPSTR lpszCmdLine, int nCmdShow)

{

wstring ws = to\_wstring(lpszCmdLine) ~ cast(wchar)0;

VSDllRegisterServerUser(ws.ptr);

}

extern (Windows)

void RunDLLUnregisterUser(HWND hwnd, HINSTANCE hinst, LPSTR lpszCmdLine, int nCmdShow)

{

wstring ws = to\_wstring(lpszCmdLine) ~ cast(wchar)0;

VSDllUnregisterServerUser(ws.ptr);

}

extern(Windows)

void VerifyMSObj(HWND hwnd, HINSTANCE hinst, LPSTR lpszCmdLine, int nCmdShow)

{

wstring ws = to\_wstring(lpszCmdLine);

VerifyMSObjectParser(ws);

}

extern(Windows)

void WritePackageDef(HWND hwnd, HINSTANCE hinst, LPSTR lpszCmdLine, int nCmdShow)

{

wstring ws = to\_wstring(lpszCmdLine) ~ cast(wchar)0;

WriteExtensionPackageDefinition(ws.ptr);

}

extern(Windows)

bool GetCoverageData(const(char)\* fname, uint line, uint\* data, uint cnt, float\* covPercent)

{

if (!Package.s\_instance)

return false; // not yet loaded as a package

string filename = to!string(fname);

return Package.GetLanguageService().GetCoverageData(filename, line, data, cnt, covPercent);

}

// This file is part of Visual D

//

// Visual D integrates the D programming language into Visual Studio

// Copyright (c) 2010 by Rainer Schuetze, All Rights Reserved

//

// Distributed under the Boost Software License, Version 1.0.

// See accompanying file LICENSE\_1\_0.txt or copy at <http://www.boost.org/LICENSE_1_0.txt>

module visuald.dpackage;

import visuald.windows;

import core.stdc.stdlib;

import std.windows.charset;

import std.string;

import std.utf;

import std.path;

import std.file;

import std.conv;

import std.array;

import std.exception;

import std.algorithm;

import stdext.path;

import stdext.array;

import stdext.file;

import stdext.string;

import stdext.registry;

import visuald.comutil;

import visuald.hierutil;

import visuald.stringutil;

import visuald.fileutil;

import visuald.dproject;

import visuald.automation;

import visuald.build;

import visuald.config;

import visuald.chiernode;

import visuald.dlangsvc;

import visuald.dimagelist;

import visuald.logutil;

import visuald.propertypage;

import visuald.winctrl;

import visuald.register;

import visuald.intellisense;

import visuald.searchsymbol;

import visuald.tokenreplacedialog;

import visuald.cppwizard;

import visuald.profiler;

import visuald.library;

import visuald.pkgutil;

import visuald.colorizer;

import visuald.dllmain;

import visuald.vdserverclient;

import sdk.win32.winreg;

import sdk.win32.oleauto;

import sdk.vsi.vsshell;

import sdk.vsi.vssplash;

import sdk.vsi.proffserv;

import sdk.vsi.vsshell90;

import sdk.vsi.objext;

import dte = sdk.vsi.dte80a;

import dte2 = sdk.vsi.dte80;

///////////////////////////////////////////////////////////////////////

struct LanguageProperty

{

wstring name;

DWORD value;

}

const string plk\_version = extractDefine(import("version"), "VERSION\_MAJOR") ~ "." ~

extractDefine(import("version"), "VERSION\_MINOR");

const string bld\_version = extractDefine(import("version"), "VERSION\_BUILD");

const string beta\_version = extractDefine(import("version"), "VERSION\_BETA");

const string full\_version = plk\_version ~ "." ~

extractDefine(import("version"), "VERSION\_REVISION") ~

(bld\_version != "0" ? beta\_version ~ bld\_version : "");

/\*---------------------------------------------------------

\* Globals

\*---------------------------------------------------------\*/

const wstring g\_languageName = "D"w;

const wstring g\_packageName = "Visual D"w;

const string g\_packageVersion = plk\_version;

const wstring g\_packageCompany = "Rainer Schuetze"w;

const wstring[] g\_languageFileExtensions = [ ".d"w, ".di"w, ".mixin"w ];

const wstring g\_projectFileExtensions = "visualdproj"w;

// CLSID registered in extensibility center (PLK)

const GUID g\_packageCLSID = uuid("002a2de9-8bb6-484d-987f-7e4ad4084715");

const GUID g\_languageCLSID = uuid("002a2de9-8bb6-484d-9800-7e4ad4084715");

const GUID g\_projectFactoryCLSID = uuid("002a2de9-8bb6-484d-9802-7e4ad4084715");

const GUID g\_intellisenseCLSID = uuid("002a2de9-8bb6-484d-9801-7e4ad4084715");

const GUID g\_commandSetCLSID = uuid("002a2de9-8bb6-484d-9803-7e4ad4084715");

const GUID g\_searchWinCLSID = uuid("002a2de9-8bb6-484d-9804-7e4ad4084715");

const GUID g\_debuggerLanguage = uuid("002a2de9-8bb6-484d-9805-7e4ad4084715");

const GUID g\_expressionEvaluator = uuid("002a2de9-8bb6-484d-9806-7e4ad4084715");

const GUID g\_profileWinCLSID = uuid("002a2de9-8bb6-484d-9807-7e4ad4084715");

const GUID g\_tokenReplaceWinCLSID = uuid("002a2de9-8bb6-484d-9808-7e4ad4084715");

const GUID g\_outputPaneCLSID = uuid("002a2de9-8bb6-484d-9809-7e4ad4084715");

const GUID g\_CppWizardWinCLSID = uuid("002a2de9-8bb6-484d-980a-7e4ad4084715");

const GUID g\_omLibraryManagerCLSID = uuid("002a2de9-8bb6-484d-980b-7e4ad4084715");

const GUID g\_omLibraryCLSID = uuid("002a2de9-8bb6-484d-980c-7e4ad4084715");

const GUID g\_ProjectItemWizardCLSID = uuid("002a2de9-8bb6-484d-980d-7e4ad4084715");

const GUID g\_unmarshalEnumOutCLSID = uuid("002a2de9-8bb6-484d-980e-7e4ad4084715");

// const GUID g\_unmarshalTargetInfoCLSID = uuid("002a2de9-8bb6-484d-980f-7e4ad4084715"); // defined in config.d

const GUID g\_VisualDHelperCLSID = uuid("002a2de9-8bb6-484d-aa10-7e4ad4084715");

// more guids in propertypage.d starting with 9810

const LanguageProperty[] g\_languageProperties =

[

// see <http://msdn.microsoft.com/en-us/library/bb166421.aspx>

{ "RequestStockColors"w, 0 },

{ "ShowCompletion"w, 1 },

{ "ShowSmartIndent"w, 1 },

{ "ShowHotURLs"w, 1 },

{ "Default to Non Hot URLs"w, 1 },

{ "DefaultToInsertSpaces"w, 0 },

{ "ShowDropdownBarOption "w, 1 },

{ "Single Code Window Only"w, 1 },

{ "EnableAdvancedMembersOption"w, 1 },

{ "Support CF\_HTML"w, 0 },

{ "EnableLineNumbersOption"w, 1 },

{ "HideAdvancedMembersByDefault"w, 0 },

];

///////////////////////////////////////////////////////////////////////

void global\_init()

{

// avoid cyclic init dependencies

initWinControls(g\_hInst);

LanguageService.shared\_static\_this();

CHierNode.shared\_static\_this();

CHierNode.shared\_static\_this\_typeHolder();

automation\_shared\_static\_this\_typeHolder();

Project.shared\_static\_this\_typeHolder();

}

void global\_exit()

{

LanguageService.shared\_static\_dtor();

CHierNode.shared\_static\_dtor\_typeHolder();

automation\_shared\_static\_dtor\_typeHolder();

Project.shared\_static\_dtor\_typeHolder();

Package.s\_instance = null;

}

///////////////////////////////////////////////////////////////////////

\_\_gshared int g\_dllRefCount;

extern(Windows)

HRESULT DllCanUnloadNow()

{

return (g\_dllRefCount == 0) ? S\_OK : S\_FALSE;

}

extern(Windows)

HRESULT DllGetClassObject(CLSID\* rclsid, IID\* riid, LPVOID\* ppv)

{

logCall("DllGetClassObject(rclsid=%s, riid=%s)", \_toLog(rclsid), \_toLog(riid));

if(\*rclsid == g\_packageCLSID)

{

auto factory = newCom!ClassFactory;

return factory.QueryInterface(riid, ppv);

}

if(\*rclsid == g\_unmarshalEnumOutCLSID)

{

DEnumOutFactory eof = newCom!DEnumOutFactory;

return eof.QueryInterface(riid, ppv);

}

static if(is(typeof(g\_unmarshalTargetInfoCLSID))) if(\*rclsid == g\_unmarshalTargetInfoCLSID)

{

TargetInfoFactory eof = newCom!TargetInfoFactory;

return eof.QueryInterface(riid, ppv);

}

if(\*rclsid == g\_ProjectItemWizardCLSID)

{

auto wiz = newCom!WizardFactory;

return wiz.QueryInterface(riid, ppv);

}

if(PropertyPageFactory factory = PropertyPageFactory.create(rclsid))

return factory.QueryInterface(riid, ppv);

return E\_NOINTERFACE;

}

///////////////////////////////////////////////////////////////////////

class ClassFactory : DComObject, IClassFactory

{

this() {}

override HRESULT QueryInterface(in IID\* riid, void\*\* pvObject)

{

if(queryInterface2!(IClassFactory) (this, IID\_IClassFactory, riid, pvObject))

return S\_OK;

return super.QueryInterface(riid, pvObject);

}

override HRESULT CreateInstance(IUnknown UnkOuter, in IID\* riid, void\*\* pvObject)

{

logCall("%s.CreateInstance(riid=%s)", this, \_toLog(riid));

if(\*riid == g\_languageCLSID)

{

assert(!UnkOuter);

LanguageService service = newCom!LanguageService(null);

return service.QueryInterface(riid, pvObject);

}

if(\*riid == IVsPackage.iid)

{

assert(!UnkOuter);

Package pkg = newCom!Package;

return pkg.QueryInterface(riid, pvObject);

}

if(\*riid == g\_unmarshalEnumOutCLSID)

{

assert(!UnkOuter);

DEnumOutputs eo = newCom!DEnumOutputs(null, 0);

return eo.QueryInterface(riid, pvObject);

}

static if(is(typeof(g\_unmarshalTargetInfoCLSID))) if(\*riid == g\_unmarshalTargetInfoCLSID)

{

assert(!UnkOuter);

auto pti = newCom!ProfilerTargetInfo(null);

return pti.QueryInterface(riid, pvObject);

}

return S\_FALSE;

}

override HRESULT LockServer(in BOOL fLock)

{

if(fLock)

InterlockedIncrement(&g\_dllRefCount);

else

InterlockedDecrement(&g\_dllRefCount);

return S\_OK;

}

int lockCount;

}

///////////////////////////////////////////////////////////////////////

static const GUID SOleComponentManager\_iid = { 0x000C060B,0x0000,0x0000,[ 0xC0,0x00,0x00,0x00,0x00,0x00,0x00,0x46 ] };

///////////////////////////////////////////////////////////////////////

class Package : DisposingComObject,

IVsPackage,

IServiceProvider,

IVsInstalledProduct,

IOleCommandTarget,

IOleComponent,

IVsPersistSolutionProps // inherits IVsPersistSolutionOpts

{

\_\_gshared Package s\_instance;

this()

{

s\_instance = this;

mOptions = new GlobalOptions();

mLangsvc = addref(newCom!LanguageService(this));

mProjFactory = addref(newCom!ProjectFactory(this));

mLibInfos = new LibraryInfos();

}

~this()

{

}

override HRESULT QueryInterface(in IID\* riid, void\*\* pvObject)

{

if(queryInterface!(IVsPackage) (this, riid, pvObject))

return S\_OK;

if(queryInterface!(IServiceProvider) (this, riid, pvObject))

return S\_OK;

if(queryInterface!(IVsInstalledProduct) (this, riid, pvObject))

return S\_OK;

if(queryInterface!(IOleCommandTarget) (this, riid, pvObject))

return S\_OK;

if(queryInterface!(IOleComponent) (this, riid, pvObject))

return S\_OK;

if(queryInterface!(IVsPersistSolutionOpts) (this, riid, pvObject))

return S\_OK;

if(queryInterface!(IVsPersistSolutionProps) (this, riid, pvObject))

return S\_OK;

return super.QueryInterface(riid, pvObject);

}

override void Dispose()

{

deleteVisualDOutputPane();

Close();

mLangsvc = release(mLangsvc);

mProjFactory = release(mProjFactory);

if(s\_instance == this)

s\_instance = null;

}

// IVsPackage

override int Close()

{

mixin(LogCallMix);

if(mHostSP)

{

CloseLibraryManager();

if(mLangServiceCookie)

{

IProfferService sc;

if(mHostSP.QueryService(&IProfferService.iid, &IProfferService.iid, cast(void\*\*)&sc) == S\_OK)

{

if(mLangServiceCookie && sc.RevokeService(mLangServiceCookie) != S\_OK)

{

OutputDebugLog("RevokeService(lang-service) failed");

}

sc.Release();

}

mLangServiceCookie = 0;

if(mLangsvc)

mLangsvc.Dispose(); // cannot call later because Package.mHostSP needed to query services

mLangsvc = release(mLangsvc);

}

if(mProjFactoryCookie)

{

IVsRegisterProjectTypes projTypes;

if(mHostSP.QueryService(&IVsRegisterProjectTypes.iid, &IVsRegisterProjectTypes.iid, cast(void\*\*)&projTypes) == S\_OK)

{

if(projTypes.UnregisterProjectType(mProjFactoryCookie) != S\_OK)

{

OutputDebugLog("UnregisterProjectType() failed");

}

projTypes.Release();

}

mProjFactoryCookie = 0;

mProjFactory = release(mProjFactory);

}

if (mComponentID != 0)

{

IOleComponentManager componentManager;

if(mHostSP.QueryService(&SOleComponentManager\_iid, &IOleComponentManager.iid, cast(void\*\*)&componentManager) == S\_OK)

{

scope(exit) release(componentManager);

componentManager.FRevokeComponent(mComponentID);

mComponentID = 0;

}

}

mHostSP = release(mHostSP);

}

return S\_OK;

}

override int CreateTool(in GUID\* rguidPersistenceSlot)

{

mixin(LogCallMix);

return E\_NOTIMPL;

}

override int GetAutomationObject(in wchar\* pszPropName, IDispatch\* ppDisp)

{

mixin(LogCallMix);

return E\_NOTIMPL;

}

override int GetPropertyPage(in GUID\* rguidPage, VSPROPSHEETPAGE\* ppage)

{

mixin(LogCallMix2);

GlobalPropertyPage tpp;

if(\*rguidPage == g\_DmdDirPropertyPage)

tpp = newCom!DmdDirPropertyPage(mOptions);

else if(\*rguidPage == g\_GdcDirPropertyPage)

tpp = newCom!GdcDirPropertyPage(mOptions);

else if(\*rguidPage == g\_LdcDirPropertyPage)

tpp = newCom!LdcDirPropertyPage(mOptions);

else if(\*rguidPage == g\_ToolsProperty2Page)

tpp = newCom!ToolsProperty2Page(mOptions);

else if(\*rguidPage == g\_ColorizerPropertyPage)

tpp = newCom!ColorizerPropertyPage(mOptions);

else if(\*rguidPage == g\_IntellisensePropertyPage)

tpp = newCom!IntellisensePropertyPage(mOptions);

else

return E\_NOTIMPL;

PROPPAGEINFO pageInfo;

pageInfo.cb = PROPPAGEINFO.sizeof;

tpp.GetPageInfo(&pageInfo);

\*ppage = VSPROPSHEETPAGE.init;

ppage.dwSize = VSPROPSHEETPAGE.sizeof;

auto win = new PropertyWindow(null, WS\_OVERLAPPED, "Visual D Settings", tpp);

win.setRect(0, 0, pageInfo.size.cx, pageInfo.size.cy);

ppage.hwndDlg = win.hwnd;

RECT r;

win.GetWindowRect(&r);

tpp.\_Activate(win, &r, false);

tpp.SetWindowSize(0, 0, pageInfo.size.cx, pageInfo.size.cy);

addref(tpp);

win.destroyDelegate = delegate void(Widget w)

{

if(auto o = tpp)

{

tpp = null;

o.Deactivate();

release(o);

}

};

win.applyDelegate = delegate void(Widget w)

{

tpp.Apply();

};

return S\_OK;

}

override int QueryClose(int\* pfCanClose)

{

mixin(LogCallMix2);

\*pfCanClose = 1;

return S\_OK;

}

override int ResetDefaults(in uint grfFlags)

{

mixin(LogCallMix);

return E\_NOTIMPL;

}

override int SetSite(IServiceProvider psp)

{

mixin(LogCallMix);

mHostSP = release(mHostSP);

mHostSP = addref(psp);

IProfferService sc;

if(mHostSP.QueryService(&IProfferService.iid, &IProfferService.iid, cast(void\*\*)&sc) == S\_OK)

{

if(sc.ProfferService(&g\_languageCLSID, this, &mLangServiceCookie) != S\_OK)

{

OutputDebugLog("ProfferService(language-service) failed");

}

sc.Release();

}

version(none)

{

// getting the debugger here causes crashes when installing/uninstalling other plugins

// command line used by installer: devenv /setup /NoSetupVSTemplates

IVsDebugger debugger;

if(mHostSP.QueryService(&IVsDebugger.iid, &IVsDebugger.iid, cast(void\*\*)&debugger) == S\_OK)

{

mLangsvc.setDebugger(debugger);

debugger.Release();

}

}

IVsRegisterProjectTypes projTypes;

if(mHostSP.QueryService(&IVsRegisterProjectTypes.iid, &IVsRegisterProjectTypes.iid, cast(void\*\*)&projTypes) == S\_OK)

{

if(projTypes.RegisterProjectType(&g\_projectFactoryCLSID, mProjFactory, &mProjFactoryCookie) != S\_OK)

{

OutputDebugLog("RegisterProjectType() failed");

}

projTypes.Release();

}

mOptions.initFromRegistry();

//register with ComponentManager for Idle processing

IOleComponentManager componentManager;

if(mHostSP.QueryService(&SOleComponentManager\_iid, &IOleComponentManager.iid, cast(void\*\*)&componentManager) == S\_OK)

{

scope(exit) release(componentManager);

if (mComponentID == 0)

{

OLECRINFO crinfo;

crinfo.cbSize = crinfo.sizeof;

crinfo.grfcrf = olecrfNeedIdleTime | olecrfNeedPeriodicIdleTime | olecrfNeedAllActiveNotifs | olecrfNeedSpecActiveNotifs;

crinfo.grfcadvf = olecadvfModal | olecadvfRedrawOff | olecadvfWarningsOff;

crinfo.uIdleTimeInterval = 1000;

if(!componentManager.FRegisterComponent(this, &crinfo, &mComponentID))

OutputDebugLog("FRegisterComponent failed");

}

}

InitLibraryManager();

return S\_OK; // E\_NOTIMPL;

}

// IServiceProvider

override int QueryService(in GUID\* guidService, in IID\* riid, void \*\* ppvObject)

{

mixin(LogCallMix);

if(mLangsvc && \*guidService == g\_languageCLSID)

return mLangsvc.QueryInterface(riid, ppvObject);

if(mProjFactory && \*guidService == g\_projectFactoryCLSID)

return mProjFactory.QueryInterface(riid, ppvObject);

return E\_NOTIMPL;

}

// IVsInstalledProduct

override int get\_IdBmpSplash(uint\* pIdBmp)

{

mixin(LogCallMix);

\*pIdBmp = BMP\_SPLASHSCRN;

return S\_OK;

}

override int get\_OfficialName(BSTR\* pbstrName)

{

logCall("%s.ProductID(pbstrName=%s)", this, pbstrName);

\*pbstrName = allocwBSTR(g\_packageName);

return S\_OK;

}

override int get\_ProductID(BSTR\* pbstrPID)

{

logCall("%s.ProductID(pbstrPID=%s)", this, pbstrPID);

\*pbstrPID = allocBSTR(full\_version);

return S\_OK;

}

override int get\_ProductDetails(BSTR\* pbstrProductDetails)

{

logCall("%s.ProductDetails(pbstrPID=%s)", this, pbstrProductDetails);

\*pbstrProductDetails = allocBSTR ("Integration of the D Programming Language into Visual Studio");

return S\_OK;

}

override int get\_IdIcoLogoForAboutbox(uint\* pIdIco)

{

logCall("%s.IdIcoLogoForAboutbox(pIdIco=%s)", this, pIdIco);

\*pIdIco = ICON\_ABOUTBOX;

return S\_OK;

}

// IOleCommandTarget //////////////////////////////////////

override int QueryStatus(in GUID \*pguidCmdGroup, in uint cCmds,

OLECMD \*prgCmds, OLECMDTEXT \*pCmdText)

{

mixin(LogCallMix);

for (uint i = 0; i < cCmds; i++)

{

if(g\_commandSetCLSID == \*pguidCmdGroup)

{

switch(prgCmds[i].cmdID)

{

case CmdSearchFile:

case CmdSearchSymbol:

case CmdSearchTokNext:

case CmdSearchTokPrev:

case CmdReplaceTokens:

case CmdConvWizard:

case CmdDustMite:

case CmdBuildPhobos:

case CmdShowProfile:

case CmdShowLangPage:

case CmdShowWebsite:

case CmdDelLstFiles:

prgCmds[i].cmdf = OLECMDF\_SUPPORTED | OLECMDF\_ENABLED;

break;

default:

break;

}

}

}

return S\_OK;

}

override int Exec( /\* [unique][in] \*/ in GUID \*pguidCmdGroup,

/\* [in] \*/ in uint nCmdID,

/\* [in] \*/ in uint nCmdexecopt,

/\* [unique][in] \*/ in VARIANT \*pvaIn,

/\* [unique][out][in] \*/ VARIANT \*pvaOut)

{

if(g\_commandSetCLSID != \*pguidCmdGroup)

return OLECMDERR\_E\_NOTSUPPORTED;

if(nCmdID == CmdSearchSymbol)

{

showSearchWindow(false);

return S\_OK;

}

if(nCmdID == CmdSearchFile)

{

showSearchWindow(true);

return S\_OK;

}

if(nCmdID == CmdSearchTokNext)

{

findNextTokenReplace(false);

return S\_OK;

}

if(nCmdID == CmdSearchTokPrev)

{

findNextTokenReplace(true);

return S\_OK;

}

if(nCmdID == CmdReplaceTokens)

{

showTokenReplaceWindow(true);

return S\_OK;

}

if(nCmdID == CmdConvWizard)

{

showCppWizardWindow();

return S\_OK;

}

if(nCmdID == CmdBuildPhobos)

{

mOptions.buildPhobosBrowseInfo();

mLibInfos.updateDefinitions();

return S\_OK;

}

if(nCmdID == CmdDustMite)

{

return DustMiteProject();

}

if(nCmdID == CmdShowProfile)

{

showProfilerWindow();

return S\_OK;

}

if(nCmdID == CmdShowLangPage)

{

auto pIVsUIShell = ComPtr!(IVsUIShell)(queryService!(IVsUIShell), false);

GUID targetGUID = uuid("734A5DE2-DEBA-11d0-A6D0-00C04FB67F6A");

VARIANT var;

var.vt = VT\_BSTR;

var.bstrVal = allocBSTR("002A2DE9-8BB6-484D-9823-7E4AD4084715");

pIVsUIShell.PostExecCommand(&CMDSETID\_StandardCommandSet97, cmdidToolsOptions, OLECMDEXECOPT\_DODEFAULT, &var);

freeBSTR(var.bstrVal);

return S\_OK;

}

if(nCmdID == CmdShowWebsite)

{

if(dte2.DTE2 spvsDTE = GetDTE())

{

scope(exit) release(spvsDTE);

spvsDTE.ExecuteCommand("View.WebBrowser"w.ptr, "[http://rainers.github.io/visuald/visuald/StartPage.html"w.ptr](http://rainers.github.io/visuald/visuald/StartPage.html%22w.ptr));

}

return S\_OK;

}

if(nCmdID == CmdDelLstFiles)

{

GetGlobalOptions().DeleteCoverageFiles();

}

return OLECMDERR\_E\_NOTSUPPORTED;

}

// IOleComponent Methods

BOOL FDoIdle(in OLEIDLEF grfidlef)

{

if(mWantsUpdateLibInfos)

{

mWantsUpdateLibInfos = false;

Package.GetLibInfos().updateDefinitions();

}

mLangsvc.OnIdle();

OutputPaneBuffer.flush();

return false;

}

void Terminate()

{

}

BOOL FPreTranslateMessage(MSG\* msg)

{

return FALSE;

}

void OnEnterState(in OLECSTATE uStateID, in BOOL fEnter)

{

}

void OnAppActivate(in BOOL fActive, in DWORD dwOtherThreadID)

{

}

void OnLoseActivation()

{

}

void OnActivationChange(/+[in]+/ IOleComponent pic,

in BOOL fSameComponent,

in const( OLECRINFO)\*pcrinfo,

in BOOL fHostIsActivating,

in const( OLECHOSTINFO)\*pchostinfo,

in DWORD dwReserved)

{

}

BOOL FReserved1(in DWORD dwReserved, in UINT message, in WPARAM wParam, in LPARAM lParam)

{

return TRUE;

}

BOOL FContinueMessageLoop(in OLELOOP uReason, in void \*pvLoopData, in MSG \*pMsgPeeked)

{

return 1;

}

BOOL FQueryTerminate( in BOOL fPromptUser)

{

return 1;

}

HWND HwndGetWindow(in OLECWINDOW dwWhich, in DWORD dwReserved)

{

return null;

}

/////////////////////////////////////////////////////////////

// IVsPersistSolutionOpts (writes to suo file)

enum slnPersistenceOpts = "VisualDProjectSolutionOptions"w;

HRESULT SaveUserOptions(IVsSolutionPersistence pPersistence)

{

mixin(LogCallMix);

return pPersistence.SavePackageUserOpts(this, slnPersistenceOpts.ptr);

}

HRESULT LoadUserOptions(IVsSolutionPersistence pPersistence, in VSLOADUSEROPTS grfLoadOpts)

{

mixin(LogCallMix);

return pPersistence.LoadPackageUserOpts(this, slnPersistenceOpts.ptr);

}

///////////////////////////

static HRESULT writeUint(IStream pStream, uint num)

{

ULONG written;

HRESULT hr = pStream.Write(&num, num.sizeof, &written);

if(hr == S\_OK && written != num.sizeof)

hr = E\_FAIL;

return hr;

}

static HRESULT writeGUID(IStream pStream, ref const GUID uid)

{

ULONG written;

HRESULT hr = pStream.Write(&uid, uid.sizeof, &written);

if(hr == S\_OK && written != uid.sizeof)

hr = E\_FAIL;

return hr;

}

static HRESULT writeString(IStream pStream, string s)

{

if(HRESULT hr = writeUint(pStream, cast(uint) s.length))

return hr;

ULONG written;

HRESULT hr = pStream.Write(s.ptr, s.length, &written);

if(hr == S\_OK && written != s.length)

hr = E\_FAIL;

return hr;

}

static HRESULT writeConfig(IStream pStream, Config cfg)

{

if(auto hr = writeString(pStream, cfg.getName()))

return hr;

if(auto hr = writeString(pStream, cfg.getPlatform()))

return hr;

xml.Document doc = xml.newDocument("SolutionOptions");

cfg.GetProjectOptions().writeDebuggerXML(doc);

string[] result = xml.writeDocument(doc);

string res = std.string.join(result, "\n");

if(auto hr = writeString(pStream, res))

return hr;

return S\_OK;

}

///////////////////////////

static HRESULT readRaw(IStream pStream, void\* p, uint size)

{

ULONG read;

HRESULT hr = pStream.Read(p, size, &read);

if(hr == S\_OK && read != size)

hr = E\_FAIL;

return hr;

}

static HRESULT readUint(IStream pStream, ref uint num)

{

return readRaw(pStream, &num, num.sizeof);

}

static HRESULT readGUID(IStream pStream, ref GUID uid)

{

return readRaw(pStream, &uid, uid.sizeof);

}

static HRESULT readString(IStream pStream, ref string s)

{

uint len;

if(HRESULT hr = readUint(pStream, len))

return hr;

if(len == -1)

return S\_FALSE;

char[] buf = new char[len];

HRESULT hr = readRaw(pStream, buf.ptr, len);

s = assumeUnique(buf);

return hr;

}

static HRESULT skip(IStream pStream, uint len)

{

char[256] buf;

for(; len >= buf.sizeof; len -= buf.sizeof)

if(auto hr = readRaw(pStream, buf.ptr, buf.sizeof))

return hr;

if(len > 0)

if(auto hr = readRaw(pStream, buf.ptr, len))

return hr;

return S\_OK;

}

HRESULT WriteUserOptions(IStream pOptionsStream, in LPCOLESTR pszKey)

{

mixin(LogCallMix);

auto srpSolution = queryService!(IVsSolution);

if(srpSolution)

{

scope(exit) release(srpSolution);

IEnumHierarchies pEnum;

if(srpSolution.GetProjectEnum(EPF\_LOADEDINSOLUTION|EPF\_MATCHTYPE, &g\_projectFactoryCLSID, &pEnum) == S\_OK)

{

scope(exit) release(pEnum);

IVsHierarchy pHierarchy;

while(pEnum.Next(1, &pHierarchy, null) == S\_OK)

{

scope(exit) release(pHierarchy);

if(IVsGetCfgProvider getCfgProvider = qi\_cast!IVsGetCfgProvider(pHierarchy))

{

scope(exit) release(getCfgProvider);

IVsCfgProvider cfgProvider;

if(getCfgProvider.GetCfgProvider(&cfgProvider) == S\_OK)

{

scope(exit) release(cfgProvider);

GUID uid;

pHierarchy.GetGuidProperty(VSITEMID\_ROOT, VSHPROPID\_ProjectIDGuid, &uid);

if(auto hr = writeGUID(pOptionsStream, uid))

return hr;

ULONG cnt;

if(cfgProvider.GetCfgs(0, null, &cnt, null) == S\_OK)

{

IVsCfg[] cfgs = new IVsCfg[cnt];

scope(exit) foreach(c; cfgs) release(c);

if(cfgProvider.GetCfgs(cnt, cfgs.ptr, &cnt, null) == S\_OK)

{

foreach(c; cfgs)

{

if(Config cfg = qi\_cast!Config(c))

{

scope(exit) release(cfg);

if(auto hr = writeConfig(pOptionsStream, cfg))

return hr;

}

}

}

}

// length -1 as end marker

if(auto hr = writeUint(pOptionsStream, -1))

return hr;

}

}

}

}

GUID uid; // empty GUID as end marker of projects

if(auto hr = writeGUID(pOptionsStream, uid))

return hr;

version(writeSearchPaneState)

{

// now followed by more chunks with (iid,length) heaser

if(auto win = getSearchPane(false))

{

if(auto hr = writeGUID(pOptionsStream, SearchPane.iid))

return hr;

if(HRESULT hr = win.SaveViewState(pOptionsStream))

return hr;

}

// empty GUID as end marker

if(auto hr = writeGUID(pOptionsStream, uid))

return hr;

}

}

return S\_OK;

}

HRESULT ReadUserOptions(IStream pOptionsStream, in LPCOLESTR pszKey)

{

                mixin(LogCallMix);

auto srpSolution = queryService!(IVsSolution);

if(!srpSolution)

return E\_FAIL;

scope(exit) release(srpSolution);

GUID uid;

for(;;)

{

if(auto hr = readGUID(pOptionsStream, uid))

return hr;

if(uid == GUID\_NULL)

break;

IVsHierarchy pHierarchy;

if (HRESULT hr = srpSolution.GetProjectOfGuid(&uid, &pHierarchy))

return hr;

scope(exit) release(pHierarchy);

IVsGetCfgProvider getCfgProvider = qi\_cast!IVsGetCfgProvider(pHierarchy);

if (!getCfgProvider)

return E\_FAIL;

scope(exit) release(getCfgProvider);

IVsCfgProvider cfgProvider;

if(auto hr = getCfgProvider.GetCfgProvider(&cfgProvider))

return hr;

scope(exit) release(cfgProvider);

IVsCfgProvider2 cfgProvider2 = qi\_cast!IVsCfgProvider2(cfgProvider);

if(!cfgProvider2)

return E\_FAIL;

scope(exit) release(cfgProvider2);

for(;;)

{

string name, platform, xmltext;

HRESULT hrName = readString(pOptionsStream, name);

if(hrName == S\_FALSE)

break;

if(hrName != S\_OK)

return hrName;

if (auto hr = readString(pOptionsStream, platform))

return hr;

if (auto hr = readString(pOptionsStream, xmltext))

return hr;

IVsCfg pCfg;

if (cfgProvider2.GetCfgOfName(\_toUTF16z(name), \_toUTF16z(platform), &pCfg) == S\_OK)

{

scope(exit) release(pCfg);

if(Config cfg = qi\_cast!Config(pCfg))

{

scope(exit) release(cfg);

try

{

xmltext = `<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>` ~ xmltext;

xml.Document doc = xml.readDocument(xmltext);

cfg.GetProjectOptions().readXML(doc);

}

catch(Exception e)

{

writeToBuildOutputPane(e.toString());

logCall(e.toString());

}

}

}

}

}

version(writeSearchPaneState)

while(readGUID(pOptionsStream, uid) == S\_OK)

{

if(uid == GUID\_NULL)

break;

if (uid == SearchPane.iid)

{

if(auto win = getSearchPane(true))

{

if(HRESULT hr = win.LoadViewState(pOptionsStream))

return hr;

continue;

}

}

// skip chunk

uint len;

if(HRESULT hr = readUint(pOptionsStream, len))

return hr;

if(HRESULT hr = skip(pOptionsStream, len))

return hr;

}

return S\_OK;

}

/////////////////////////////////////////////////////////////

// IVsPersistSolutionProps (writes to sln file)

enum slnPersistenceKey = "VisualDProjectSolutionProperties"w;

enum slnPersistenceValue = "TestValue"w;

override HRESULT QuerySaveSolutionProps(IVsHierarchy pHierarchy, VSQUERYSAVESLNPROPS \*pqsspSave)

{

mixin(LogCallMix);

Project prj = qi\_cast!Project(pHierarchy);

if(!prj)

return E\_NOINTERFACE;

release(prj);

\*pqsspSave = QSP\_HasNoProps;

return S\_OK;

}

override HRESULT SaveSolutionProps(IVsHierarchy pHierarchy, IVsSolutionPersistence pPersistence)

{

mixin(LogCallMix);

return pPersistence.SavePackageSolutionProps(false, pHierarchy, this, slnPersistenceKey.ptr);

}

override HRESULT WriteSolutionProps(IVsHierarchy pHierarchy, in LPCOLESTR pszKey, IPropertyBag pPropBag)

{

mixin(LogCallMix);

Project prj = qi\_cast!Project(pHierarchy);

if(!prj)

return E\_NOINTERFACE;

release(prj);

version(none)

{

VARIANT var;

var.vt = VT\_BSTR;

var.bstrVal = allocBSTR("Test");

HRESULT hr = pPropBag.Write(slnPersistenceValue.ptr, &var);

freeBSTR(var.bstrVal);

}

return S\_OK;

}

override HRESULT ReadSolutionProps(IVsHierarchy pHierarchy, in LPCOLESTR pszProjectName,

in LPCOLESTR pszProjectMk, in LPCOLESTR pszKey,

in BOOL fPreLoad, /+[in]+/ IPropertyBag pPropBag)

{

mixin(LogCallMix);

if(slnPersistenceKey == to\_wstring(pszKey))

{

VARIANT var;

if(pPropBag.Read(slnPersistenceValue.ptr, &var, null) == S\_OK)

{

if (var.vt == VT\_BSTR)

{

string value = detachBSTR(var.bstrVal);

}

}

}

return S\_OK;

}

override HRESULT OnProjectLoadFailure(IVsHierarchy pStubHierarchy, in LPCOLESTR pszProjectName,

in LPCOLESTR pszProjectMk, in LPCOLESTR pszKey)

{

mixin(LogCallMix);

return S\_OK;

}

/////////////////////////////////////////////////////////////

HRESULT InitLibraryManager()

{

if (mOmLibraryCookie != 0) // already init-ed

return E\_UNEXPECTED;

HRESULT hr = E\_FAIL;

if(auto om = queryService!(IVsObjectManager, IVsObjectManager2))

{

scope(exit) release(om);

mLibrary = newCom!Library;

hr = om.RegisterSimpleLibrary(mLibrary, &mOmLibraryCookie);

if(SUCCEEDED(hr))

mLibrary.Initialize();

}

return hr;

}

HRESULT CloseLibraryManager()

{

if (mOmLibraryCookie == 0) // already closed or not init-ed

return S\_OK;

HRESULT hr = E\_FAIL;

if(auto om = queryService!(IVsObjectManager, IVsObjectManager2))

{

scope(exit) release(om);

hr = om.UnregisterLibrary(mOmLibraryCookie);

mLibrary.Close(); // attaches itself to SolutionEvents, so we need to break circular reference

mLibrary = null;

}

mOmLibraryCookie = 0;

return hr;

}

/////////////////////////////////////////////////////////////

IServiceProvider getServiceProvider()

{

return mHostSP;

}

static LanguageService GetLanguageService()

{

assert(s\_instance);

return s\_instance.mLangsvc;

}

static GlobalOptions GetGlobalOptions()

{

assert(s\_instance);

return s\_instance.mOptions;

}

static LibraryInfos GetLibInfos()

{

assert(s\_instance);

return s\_instance.mLibInfos;

}

static Library GetLibrary()

{

assert(s\_instance);

return s\_instance.mLibrary;

}

static void scheduleUpdateLibrary()

{

assert(s\_instance);

s\_instance.mWantsUpdateLibInfos = true;

}

private:

IServiceProvider mHostSP;

uint mLangServiceCookie;

uint mProjFactoryCookie;

uint mComponentID;

LanguageService mLangsvc;

ProjectFactory mProjFactory;

uint mOmLibraryCookie;

GlobalOptions mOptions;

LibraryInfos mLibInfos;

bool mWantsUpdateLibInfos;

Library mLibrary;

}

struct CompilerDirectories

{

string InstallDir;

string ExeSearchPath;

string ImpSearchPath;

string LibSearchPath;

string DisasmCommand;

string ExeSearchPath64;

string LibSearchPath64;

bool overrideIni64;

string overrideLinker64;

string overrideOptions64;

string DisasmCommand64;

string ExeSearchPath32coff;

string LibSearchPath32coff;

bool overrideIni32coff;

string overrideLinker32coff;

string overrideOptions32coff;

string DisasmCommand32coff;

}

class GlobalOptions

{

HKEY hConfigKey;

HKEY hUserKey;

wstring regConfigRoot;

wstring regUserRoot;

CompilerDirectories DMD;

CompilerDirectories GDC;

CompilerDirectories LDC;

string IncSearchPath;

string JSNSearchPath;

string UserTypesSpec;

int[wstring] UserTypes;

// evaluated once at startup

string WindowsSdkDir;

string UCRTSdkDir;

string UCRTVersion;

string DevEnvDir;

string VSInstallDir;

string VCInstallDir;

string VisualDInstallDir;

bool timeBuilds;

bool sortProjects = true;

bool stopSolutionBuild;

bool showUptodateFailure;

bool demangleError = true;

bool optlinkDeps = true;

bool autoOutlining;

byte deleteFiles; // 0: ask, -1: don't delete, 1: delete (obsolete)

bool parseSource;

bool pasteIndent;

bool expandFromSemantics;

bool expandFromBuffer;

bool expandFromJSON;

byte expandTrigger;

bool showTypeInTooltip;

bool semanticGotoDef;

bool useDParser;

bool mixinAnalysis;

bool UFCSExpansions;

string VDServerIID;

string compileAndRunOpts;

string compileAndDbgOpts;

int compileAndDbgEngine;

string[] coverageBuildDirs;

string[] coverageExecutionDirs;

bool showCoverageMargin;

bool ColorizeCoverage = true;

bool ColorizeVersions = true;

bool lastColorizeCoverage;

bool lastColorizeVersions;

bool lastUseDParser;

this()

{

}

bool getRegistryRoot()

{

if(hConfigKey)

return true;

BSTR bstrRoot;

ILocalRegistry4 registry4 = queryService!(ILocalRegistry, ILocalRegistry4);

if(registry4)

{

scope(exit) release(registry4);

if(registry4.GetLocalRegistryRootEx(RegType\_Configuration, cast(uint\*)&hConfigKey, &bstrRoot) == S\_OK)

{

regConfigRoot = wdetachBSTR(bstrRoot);

if(registry4.GetLocalRegistryRootEx(RegType\_UserSettings, cast(uint\*)&hUserKey, &bstrRoot) == S\_OK)

regUserRoot = wdetachBSTR(bstrRoot);

else

{

regUserRoot = regConfigRoot;

hUserKey = HKEY\_CURRENT\_USER;

}

return true;

}

}

ILocalRegistry2 registry = queryService!(ILocalRegistry, ILocalRegistry2);

if(registry)

{

scope(exit) release(registry);

if(registry.GetLocalRegistryRoot(&bstrRoot) == S\_OK)

{

regConfigRoot = wdetachBSTR(bstrRoot);

hConfigKey = HKEY\_LOCAL\_MACHINE;

regUserRoot = regConfigRoot;

hUserKey = HKEY\_CURRENT\_USER;

return true;

}

}

return false;

}

void detectWindowsSDKDir()

{

// todo: detect Win10 SDK

if(WindowsSdkDir.empty)

{

scope RegKey keySdk = new RegKey(HKEY\_LOCAL\_MACHINE, "SOFTWARE\\Microsoft\\Microsoft SDKs\\Windows\\v8.1"w, false);

WindowsSdkDir = toUTF8(keySdk.GetString("InstallationFolder"));

if(!std.file.exists(buildPath(WindowsSdkDir, "Lib")))

WindowsSdkDir = "";

}

if(WindowsSdkDir.empty)

{

scope RegKey keySdk = new RegKey(HKEY\_LOCAL\_MACHINE, "SOFTWARE\\Microsoft\\Microsoft SDKs\\Windows\\v8.0"w, false);

WindowsSdkDir = toUTF8(keySdk.GetString("InstallationFolder"));

if(!std.file.exists(buildPath(WindowsSdkDir, "Lib")))

WindowsSdkDir = "";

}

if(WindowsSdkDir.empty)

{

scope RegKey keySdk = new RegKey(HKEY\_LOCAL\_MACHINE, "SOFTWARE\\Microsoft\\Microsoft SDKs\\Windows"w, false);

WindowsSdkDir = toUTF8(keySdk.GetString("CurrentInstallFolder"));

if(!std.file.exists(buildPath(WindowsSdkDir, "Lib")))

WindowsSdkDir = "";

}

if(WindowsSdkDir.empty)

if(char\* psdk = getenv("WindowsSdkDir"))

WindowsSdkDir = fromMBSz(cast(immutable)psdk);

if(!WindowsSdkDir.empty)

WindowsSdkDir = normalizeDir(WindowsSdkDir);

}

void detectUCRT()

{

if(UCRTSdkDir.empty)

{

if(char\* psdk = getenv("UniversalCRTSdkDir"))

UCRTSdkDir = normalizeDir(fromMBSz(cast(immutable)psdk));

else

{

scope RegKey keySdk = new RegKey(HKEY\_LOCAL\_MACHINE, "SOFTWARE\\Microsoft\\Windows Kits\\Installed Roots"w, false);

UCRTSdkDir = normalizeDir(toUTF8(keySdk.GetString("KitsRoot10")));

}

}

if(UCRTVersion.empty)

{

if(char\* pver = getenv("UCRTVersion"))

UCRTVersion = fromMBSz(cast(immutable)pver);

else if(!UCRTSdkDir.empty)

{

string rootsDir = normalizeDir(UCRTSdkDir) ~ "Lib\\";

try

{

foreach(string f; dirEntries(rootsDir, "\*", SpanMode.shallow, false))

if(std.file.isDir(f) && f > UCRTVersion)

UCRTVersion = baseName(f);

}

catch(Exception)

{

}

}

}

}

void detectVCInstallDir()

{

if(char\* pe = getenv("VSINSTALLDIR"))

VSInstallDir = fromMBSz(cast(immutable)pe);

else

{

scope RegKey keyVS = new RegKey(hConfigKey, regConfigRoot, false);

VSInstallDir = toUTF8(keyVS.GetString("InstallDir"));

// InstallDir is ../Common7/IDE/

VSInstallDir = normalizeDir(VSInstallDir);

VSInstallDir = dirName(dirName(VSInstallDir));

}

VSInstallDir = normalizeDir(VSInstallDir);

}

void detectVSInstallDir()

{

if(char\* pe = getenv("VCINSTALLDIR"))

VCInstallDir = fromMBSz(cast(immutable)pe);

else

{

scope RegKey keyVS = new RegKey(hConfigKey, regConfigRoot ~ "[\\Setup\\VC](file:///\\Setup\VC)", false);

VCInstallDir = toUTF8(keyVS.GetString("ProductDir"));

}

VCInstallDir = normalizeDir(VCInstallDir);

}

bool initFromRegistry()

{

if(!getRegistryRoot())

return false;

wstring dllPath = GetDLLName(g\_hInst);

VisualDInstallDir = normalizeDir(dirName(toUTF8(dllPath)));

wstring idePath = GetDLLName(null);

DevEnvDir = normalizeDir(dirName(toUTF8(idePath)));

bool rc = true;

try

{

wstring defUserTypesSpec = "Object string wstring dstring ClassInfo\n"

"hash\_t ptrdiff\_t size\_t sizediff\_t";

// get defaults from global config

scope RegKey keyToolOpts = new RegKey(hConfigKey, regConfigRoot ~ regPathToolsOptions, false);

scope RegKey keyUserOpts = new RegKey(hUserKey, regUserRoot ~ regPathToolsOptions, false);

detectWindowsSDKDir();

detectUCRT();

detectVSInstallDir();

detectVCInstallDir();

wstring getWStringOpt(wstring tag, wstring def = null)

{

wstring ws = keyToolOpts.GetString(tag, def);

return keyUserOpts.GetString(tag, ws);

}

string getStringOpt(wstring tag, wstring def = null)

{

return toUTF8(getWStringOpt(tag, def));

}

string getPathsOpt(wstring tag, string def = null)

{

return replaceSemiCrLf(toUTF8(getWStringOpt(tag, to!wstring(def))));

}

int getIntOpt(wstring tag, int def = 0)

{

int v = keyToolOpts.GetDWORD(tag, def);

return keyUserOpts.GetDWORD(tag, v);

}

bool getBoolOpt(wstring tag, bool def = false)

{

return getIntOpt(tag, def ? 1 : 0) != 0;

}

ColorizeVersions = getBoolOpt("ColorizeVersions", true);

ColorizeCoverage = getBoolOpt("ColorizeCoverage", true);

showCoverageMargin = getBoolOpt("showCoverageMargin", false);

timeBuilds = getBoolOpt("timeBuilds", false);

sortProjects = getBoolOpt("sortProjects", true);

stopSolutionBuild = getBoolOpt("stopSolutionBuild", false);

showUptodateFailure = getBoolOpt("showUptodateFailure", false);

demangleError = getBoolOpt("demangleError", true);

optlinkDeps = getBoolOpt("optlinkDeps", true);

autoOutlining = getBoolOpt("autoOutlining", true);

deleteFiles = cast(byte) getIntOpt("deleteFiles", 0);

parseSource = getBoolOpt("parseSource", true);

expandFromSemantics = getBoolOpt("expandFromSemantics", true);

expandFromBuffer = getBoolOpt("expandFromBuffer", true);

expandFromJSON = getBoolOpt("expandFromJSON", true);

expandTrigger = cast(byte) getIntOpt("expandTrigger", 0);

showTypeInTooltip = getBoolOpt("showTypeInTooltip2", true); // changed default

semanticGotoDef = getBoolOpt("semanticGotoDef", true);

pasteIndent = getBoolOpt("pasteIndent", true);

scope RegKey keyDParser = new RegKey(HKEY\_CLASSES\_ROOT, "CLSID\\{002a2de9-8bb6-484d-AA05-7e4ad4084715}", false);

useDParser = true; // getBoolOpt("useDParser2", keyDParser.key !is null);

mixinAnalysis = getBoolOpt("mixinAnalysis", false);

UFCSExpansions = getBoolOpt("UFCSExpansions", true);

string getDefaultLibPathCOFF64()

{

string libpath = r"$(VCInstallDir)\lib\amd64";

if(std.file.exists(VCInstallDir ~ "lib\\legacy\_stdio\_definitions.lib"))

libpath ~= "\n$(UCRTSdkDir)Lib\\$(UCRTVersion)\\ucrt\\x64";

if(WindowsSdkDir.length)

{

if(std.file.exists(WindowsSdkDir ~ r"lib\x64\kernel32.lib"))

libpath ~= "\n$(WindowsSdkDir)lib\\x64";

else if(std.file.exists(WindowsSdkDir ~ r"Lib\win8\um\x64\kernel32.lib")) // SDK 8.0

libpath ~= "\n$(WindowsSdkDir)Lib\\win8\\um\\x64";

else if(std.file.exists(WindowsSdkDir ~ r"Lib\winv6.3\um\x64\kernel32.lib")) // SDK 8.1

libpath ~= "\n$(WindowsSdkDir)Lib\\winv6.3\\um\\x64";

}

return libpath;

}

string getDefaultLibPathCOFF32()

{

string libpath = r"$(VCInstallDir)\lib";

if(std.file.exists(VCInstallDir ~ "lib\\legacy\_stdio\_definitions.lib"))

libpath ~= "\n$(UCRTSdkDir)Lib\\$(UCRTVersion)\\ucrt\\x86";

if(WindowsSdkDir.length)

{

if(std.file.exists(WindowsSdkDir ~ r"lib\kernel32.lib"))

libpath ~= "\n$(WindowsSdkDir)lib";

else if(std.file.exists(WindowsSdkDir ~ r"Lib\win8\um\x86\kernel32.lib")) // SDK 8.0

libpath ~= "\n$(WindowsSdkDir)Lib\\win8\\um\\x86";

else if(std.file.exists(WindowsSdkDir ~ r"Lib\winv6.3\um\x86\kernel32.lib")) // SDK 8.1

libpath ~= "\n$(WindowsSdkDir)Lib\\winv6.3\\um\\x86";

}

return libpath;

}

// overwrite by user config

void readCompilerOptions(string compiler)(ref CompilerDirectories opt)

{

enum bool dmd = compiler == "DMD";

enum string prefix = dmd ? "" : compiler ~ ".";

opt.InstallDir = getStringOpt(compiler ~ "InstallDir");

opt.ExeSearchPath = getPathsOpt(prefix ~ "ExeSearchPath", opt.ExeSearchPath);

opt.LibSearchPath = getPathsOpt(prefix ~ "LibSearchPath", opt.LibSearchPath);

opt.ImpSearchPath = getPathsOpt(prefix ~ "ImpSearchPath", opt.ImpSearchPath);

opt.DisasmCommand = getPathsOpt(prefix ~ "DisasmCommand", opt.DisasmCommand);

opt.ExeSearchPath64 = getPathsOpt(prefix ~ "ExeSearchPath64", opt.ExeSearchPath64);

opt.LibSearchPath64 = getPathsOpt(prefix ~ "LibSearchPath64", opt.LibSearchPath64);

opt.DisasmCommand64 = getPathsOpt(prefix ~ "DisasmCommand64", opt.DisasmCommand64);

opt.overrideIni64 = getBoolOpt(prefix ~ "overrideIni64", dmd);

opt.overrideLinker64 = getStringOpt(prefix ~ "overrideLinker64", dmd ? r"$(VCINSTALLDIR)\bin\link.exe" : "");

opt.overrideOptions64 = getStringOpt(prefix ~ "overrideOptions64");

if (dmd)

{

opt.ExeSearchPath32coff = getPathsOpt(prefix ~ "ExeSearchPath32coff", opt.ExeSearchPath32coff);

opt.LibSearchPath32coff = getPathsOpt(prefix ~ "LibSearchPath32coff", opt.LibSearchPath32coff);

opt.DisasmCommand32coff = getPathsOpt(prefix ~ "DisasmCommand32coff", opt.DisasmCommand32coff);

opt.overrideIni32coff = getBoolOpt(prefix ~ "overrideIni32coff", dmd);

opt.overrideLinker32coff = getStringOpt(prefix ~ "overrideLinker32coff", dmd ? r"$(VCINSTALLDIR)\bin\link.exe" : "");

opt.overrideOptions32coff = getStringOpt(prefix ~ "overrideOptions32coff");

}

}

// put dmd bin folder at the end to avoid trouble with link.exe (dmd does not need search path)

// $(WindowsSdkDir)\bin needed for rc.exe

// $(VCInstallDir)\bin needed to compile C + link.exe + DLLs

// $(VSINSTALLDIR)\Common7\IDE needed for some VS versions for cv2pdb

DMD.ExeSearchPath = r"$(VCInstallDir)\bin;$(VSINSTALLDIR)\Common7\IDE;$(WindowsSdkDir)\bin;$(DMDInstallDir)windows\bin";

DMD.ExeSearchPath64 = DMD.ExeSearchPath;

DMD.ExeSearchPath32coff = DMD.ExeSearchPath;

GDC.ExeSearchPath = r"$(GDCInstallDir)\bin;$(VSINSTALLDIR)\Common7\IDE;$(WindowsSdkDir)\bin";

GDC.ExeSearchPath64 = GDC.ExeSearchPath;

LDC.ExeSearchPath = r"$(LDCInstallDir)\bin;$(VCInstallDir)\bin;$(VSINSTALLDIR)\Common7\IDE;$(WindowsSdkDir)\bin";

LDC.ExeSearchPath64 = r"$(LDCInstallDir)\bin;$(VCInstallDir)\bin\amd64;$(WindowsSdkDir)\bin";

DMD.LibSearchPath64 = getDefaultLibPathCOFF64();

LDC.LibSearchPath64 = DMD.LibSearchPath64;

DMD.LibSearchPath32coff = getDefaultLibPathCOFF32();

LDC.LibSearchPath = DMD.LibSearchPath32coff;

DMD.DisasmCommand = `"obj2asm" -x "$(InputPath)" >"$(TargetPath)"`;

DMD.DisasmCommand64 = `"$(VCInstallDir)\bin\amd64\dumpbin" /disasm:nobytes "$(InputPath)" >"$(TargetPath)"`;

DMD.DisasmCommand32coff = `"$(VCInstallDir)\bin\dumpbin" /disasm:nobytes "$(InputPath)" >"$(TargetPath)"`;

GDC.DisasmCommand = DMD.DisasmCommand32coff;

LDC.DisasmCommand = DMD.DisasmCommand32coff;

GDC.DisasmCommand64 = DMD.DisasmCommand64;

LDC.DisasmCommand64 = DMD.DisasmCommand64;

readCompilerOptions!"DMD"(DMD);

readCompilerOptions!"GDC"(GDC);

readCompilerOptions!"LDC"(LDC);

JSNSearchPath = getPathsOpt("JSNSearchPath");

IncSearchPath = getStringOpt("IncSearchPath", r"$(WindowsSdkDir)\include;$(VCInstallDir)\include");

VDServerIID = getStringOpt("VDServerIID");

compileAndRunOpts = getStringOpt("compileAndRunOpts", "-unittest");

compileAndDbgOpts = getStringOpt("compileAndDbgOpts", "-g");

compileAndDbgEngine = getIntOpt("compileAndDbgEngine", 0);

string execDirs = getStringOpt("coverageExecutionDirs", "");

coverageExecutionDirs = split(execDirs, ";");

string buildDirs = getStringOpt("coverageBuildDirs", "");

coverageBuildDirs = split(buildDirs, ";");

UserTypesSpec = getStringOpt("UserTypesSpec", defUserTypesSpec);

UserTypes = parseUserTypes(UserTypesSpec);

lastColorizeCoverage = ColorizeCoverage;

lastColorizeVersions = ColorizeVersions;

lastUseDParser = useDParser;

updateDefaultColors();

if(VDServerIID.length > 0)

gServerClassFactory\_iid = uuid(VDServerIID);

else

updateVDServer();

CHierNode.setContainerIsSorted(sortProjects);

}

catch(Exception e)

{

writeToBuildOutputPane(e.msg);

rc = false;

}

return rc;

}

void updateVDServer()

{

if(useDParser)

gServerClassFactory\_iid = DParserClassFactory\_iid;

else

gServerClassFactory\_iid = VDServerClassFactory\_iid;

}

bool saveToRegistry()

{

if(!getRegistryRoot())

return false;

try

{

scope RegKey keyToolOpts = new RegKey(hUserKey, regUserRoot ~ regPathToolsOptions);

keyToolOpts.Set("DMDInstallDir", toUTF16(DMD.InstallDir));

keyToolOpts.Set("GDCInstallDir", toUTF16(GDC.InstallDir));

keyToolOpts.Set("LDCInstallDir", toUTF16(LDC.InstallDir));

keyToolOpts.Set("ExeSearchPath", toUTF16(DMD.ExeSearchPath));

keyToolOpts.Set("LibSearchPath", toUTF16(DMD.LibSearchPath));

keyToolOpts.Set("ImpSearchPath", toUTF16(DMD.ImpSearchPath));

keyToolOpts.Set("DisasmCommand", toUTF16(DMD.DisasmCommand));

keyToolOpts.Set("GDC.ExeSearchPath", toUTF16(GDC.ExeSearchPath));

keyToolOpts.Set("GDC.LibSearchPath", toUTF16(GDC.LibSearchPath));

keyToolOpts.Set("GDC.ImpSearchPath", toUTF16(GDC.ImpSearchPath));

keyToolOpts.Set("GDC.DisasmCommand", toUTF16(GDC.DisasmCommand));

keyToolOpts.Set("LDC.ExeSearchPath", toUTF16(LDC.ExeSearchPath));

keyToolOpts.Set("LDC.LibSearchPath", toUTF16(LDC.LibSearchPath));

keyToolOpts.Set("LDC.ImpSearchPath", toUTF16(LDC.ImpSearchPath));

keyToolOpts.Set("LDC.DisasmCommand", toUTF16(LDC.DisasmCommand));

keyToolOpts.Set("JSNSearchPath", toUTF16(JSNSearchPath));

keyToolOpts.Set("IncSearchPath", toUTF16(IncSearchPath));

keyToolOpts.Set("UserTypesSpec", toUTF16(UserTypesSpec));

keyToolOpts.Set("ExeSearchPath64", toUTF16(DMD.ExeSearchPath64));

keyToolOpts.Set("LibSearchPath64", toUTF16(DMD.LibSearchPath64));

keyToolOpts.Set("DisasmCommand64", toUTF16(DMD.DisasmCommand64));

keyToolOpts.Set("overrideIni64", DMD.overrideIni64);

keyToolOpts.Set("overrideLinker64", toUTF16(DMD.overrideLinker64));

keyToolOpts.Set("overrideOptions64", toUTF16(DMD.overrideOptions64));

keyToolOpts.Set("ExeSearchPath32coff", toUTF16(DMD.ExeSearchPath32coff));

keyToolOpts.Set("LibSearchPath32coff", toUTF16(DMD.LibSearchPath32coff));

keyToolOpts.Set("DisasmCommand32coff", toUTF16(DMD.DisasmCommand32coff));

keyToolOpts.Set("overrideIni32coff", DMD.overrideIni32coff);

keyToolOpts.Set("overrideLinker32coff", toUTF16(DMD.overrideLinker32coff));

keyToolOpts.Set("overrideOptions32coff", toUTF16(DMD.overrideOptions32coff));

keyToolOpts.Set("GDC.ExeSearchPath64", toUTF16(GDC.ExeSearchPath64));

keyToolOpts.Set("GDC.LibSearchPath64", toUTF16(GDC.LibSearchPath64));

keyToolOpts.Set("GDC.DisasmCommand64", toUTF16(GDC.DisasmCommand64));

keyToolOpts.Set("LDC.ExeSearchPath64", toUTF16(LDC.ExeSearchPath64));

keyToolOpts.Set("LDC.LibSearchPath64", toUTF16(LDC.LibSearchPath64));

keyToolOpts.Set("LDC.DisasmCommand64", toUTF16(LDC.DisasmCommand64));

keyToolOpts.Set("ColorizeVersions", ColorizeVersions);

keyToolOpts.Set("ColorizeCoverage", ColorizeCoverage);

keyToolOpts.Set("showCoverageMargin", showCoverageMargin);

keyToolOpts.Set("timeBuilds", timeBuilds);

keyToolOpts.Set("sortProjects", sortProjects);

keyToolOpts.Set("stopSolutionBuild", stopSolutionBuild);

keyToolOpts.Set("showUptodateFailure", showUptodateFailure);

keyToolOpts.Set("demangleError", demangleError);

keyToolOpts.Set("optlinkDeps", optlinkDeps);

keyToolOpts.Set("autoOutlining", autoOutlining);

keyToolOpts.Set("deleteFiles", deleteFiles);

keyToolOpts.Set("parseSource", parseSource);

keyToolOpts.Set("expandFromSemantics", expandFromSemantics);

keyToolOpts.Set("expandFromBuffer", expandFromBuffer);

keyToolOpts.Set("expandFromJSON", expandFromJSON);

keyToolOpts.Set("expandTrigger", expandTrigger);

keyToolOpts.Set("showTypeInTooltip2", showTypeInTooltip);

keyToolOpts.Set("semanticGotoDef", semanticGotoDef);

keyToolOpts.Set("useDParser2", useDParser);

keyToolOpts.Set("mixinAnalysis", mixinAnalysis);

keyToolOpts.Set("UFCSExpansions", UFCSExpansions);

keyToolOpts.Set("pasteIndent", pasteIndent);

keyToolOpts.Set("compileAndRunOpts", toUTF16(compileAndRunOpts));

keyToolOpts.Set("compileAndDbgOpts", toUTF16(compileAndDbgOpts));

keyToolOpts.Set("compileAndDbgEngine", compileAndDbgEngine);

keyToolOpts.Set("coverageExecutionDirs", toUTF16(join(coverageExecutionDirs, ";")));

keyToolOpts.Set("coverageBuildDirs", toUTF16(join(coverageBuildDirs, ";")));

CHierNode.setContainerIsSorted(sortProjects);

}

catch(Exception e)

{

writeToBuildOutputPane(e.msg);

return false;

}

bool updateColorizer = false;

int[wstring] types = parseUserTypes(UserTypesSpec);

if(types != UserTypes)

{

UserTypes = types;

updateColorizer = true;

}

if(lastColorizeVersions != ColorizeVersions)

{

lastColorizeVersions = ColorizeVersions;

updateColorizer = true;

}

if(lastColorizeCoverage != ColorizeCoverage)

{

lastColorizeCoverage = ColorizeCoverage;

updateColorizer = true;

}

if(updateColorizer)

if(auto svc = Package.s\_instance.mLangsvc)

svc.UpdateColorizer(true);

if(lastUseDParser != useDParser)

{

updateVDServer();

lastUseDParser = useDParser;

VDServerClient.restartServer = true;

}

else if(!expandFromSemantics)

Package.GetLanguageService().ClearSemanticProject();

Package.scheduleUpdateLibrary();

return true;

}

void addReplacements(ref string[string] replacements)

{

replacements["DMDINSTALLDIR"] = normalizeDir(DMD.InstallDir);

replacements["GDCINSTALLDIR"] = normalizeDir(GDC.InstallDir);

replacements["LDCINSTALLDIR"] = normalizeDir(LDC.InstallDir);

replacements["WINDOWSSDKDIR"] = WindowsSdkDir;

replacements["UCRTSDKDIR"] = UCRTSdkDir;

replacements["UCRTVERSION"] = UCRTVersion;

replacements["DEVENVDIR"] = DevEnvDir;

replacements["VCINSTALLDIR"] = VCInstallDir;

replacements["VSINSTALLDIR"] = VSInstallDir;

replacements["VISUALDINSTALLDIR"] = VisualDInstallDir;

}

string replaceGlobalMacros(string s)

{

if(s.indexOf('$') < 0)

return s;

string[string] replacements;

addReplacements(replacements);

return replaceMacros(s, replacements);

}

string findInPath(string exe)

{

string searchpaths = replaceGlobalMacros(DMD.ExeSearchPath);

string[] paths = tokenizeArgs(searchpaths, true, false);

if(char\* p = getenv("PATH"))

paths ~= tokenizeArgs(to!string(p), true, false);

foreach(path; paths)

{

path = unquoteArgument(path);

path = normalizeDir(path);

if(std.file.exists(path ~ exe))

return path;

}

return null;

}

string findDmdBinDir(string dmdpath = null)

{

if(dmdpath.length && std.file.exists(dmdpath))

return normalizeDir(dirName(dmdpath));

string installdir = normalizeDir(DMD.InstallDir);

string bindir = installdir ~ "windows\\bin\\";

if(std.file.exists(bindir ~ "dmd.exe"))

return bindir;

string dmd = findInPath("dmd.exe");

return empty(dmd) ? null : dirName(dmd);

}

string findScIni(string workdir, string dmdpath, bool optlink)

{

string inifile;

if(workdir.length)

inifile = buildPath(workdir, "sc.ini");

if(inifile.empty || !std.file.exists(inifile))

{

inifile = null;

if(auto home = getenv("HOME"))

inifile = buildPath(fromMBSz(cast(immutable)home), "sc.ini");

}

if(inifile.empty || !std.file.exists(inifile))

{

inifile = null;

string dmddir = findDmdBinDir(dmdpath);

if(!dmddir.empty)

{

if(optlink)

dmddir = dmddir; // TODO: in case link is elsewhere it uses a different sc.ini

inifile = buildPath(dmddir, "sc.ini");

}

}

if(inifile.empty || !std.file.exists(inifile))

inifile = null;

return inifile;

}

string getLinkerPath(bool x64, bool mscoff, string workdir, string dmdpath, string \*libs = null, string\* options = null)

{

string path = "link.exe";

string inifile = findScIni(workdir, dmdpath, false);

if(!inifile.empty)

{

string[string] env = [ "@P" : dirName(inifile) ];

addReplacements(env);

string[string][string] ini = parseIni(inifile);

if(auto pEnv = "Environment" in ini)

env = expandIniSectionEnvironment((\*pEnv)[""], env);

string envArch = x64 ? "Environment64" : mscoff ? "Environment32mscoff" : "Environment32";

if(auto pEnv = envArch in ini)

env = expandIniSectionEnvironment((\*pEnv)[""], env);

if(string\* pLink = "LINKCMD" in env)

path = \*pLink;

if(x64)

{

if(DMD.overrideIni64)

path = DMD.overrideLinker64;

else if(string\* pLink = "LINKCMD64" in env)

path = \*pLink;

}

else if(mscoff)

{

if(DMD.overrideIni32coff)

path = DMD.overrideLinker32coff;

}

if(options)

{

if(x64 && DMD.overrideIni64)

\*options = DMD.overrideOptions64;

else if(!x64 && mscoff && DMD.overrideIni32coff)

\*options = DMD.overrideOptions32coff;

else if(string\* pFlags = "DFLAGS" in env)

\*options = \*pFlags;

}

if(libs)

if(string\* pLibs = "LIB" in env)

\*libs = \*pLibs;

}

return path;

}

static string[] getOptionImportPaths(string opts, string workdir)

{

string[] imports;

string[] args = tokenizeArgs(opts);

args = expandResponseFiles(args, workdir);

foreach(arg; args)

{

arg = unquoteArgument(arg);

if(arg.startsWith("-I"))

imports ~= removeDotDotPath(normalizeDir(arg[2..$]));

}

return imports;

}

string[] getIniImportPaths()

{

string[] imports;

string bindir = findDmdBinDir();

string inifile = bindir ~ "sc.ini";

if(std.file.exists(inifile))

{

string[string][string] ini = parseIni(inifile);

if(auto pEnv = "Environment" in ini)

if(string\* pFlags = "DFLAGS" in \*pEnv)

{

string opts = replace(\*pFlags, "%@P%", bindir);

imports ~= getOptionImportPaths(opts, bindir);

}

}

return imports;

}

string[] getImportPaths()

{

string[] imports = getIniImportPaths();

string searchpaths = replaceGlobalMacros(DMD.ImpSearchPath);

string[] args = tokenizeArgs(searchpaths);

foreach(arg; args)

imports ~= removeDotDotPath(normalizeDir(unquoteArgument(arg)));

return imports;

}

string[] getJSONPaths()

{

string[] jsonpaths;

string searchpaths = replaceGlobalMacros(JSNSearchPath);

string[] args = tokenizeArgs(searchpaths);

foreach(arg; args)

jsonpaths ~= normalizeDir(unquoteArgument(arg));

return jsonpaths;

}

string[] getJSONFiles()

{

string[] jsonpaths = getJSONPaths();

string[] jsonfiles;

foreach(path; jsonpaths)

{

if(isExistingDir(path))

foreach (string name; dirEntries(path, SpanMode.shallow))

if (globMatch(baseName(name), "\*.json"))

addunique(jsonfiles, name);

}

return jsonfiles;

}

void logSettingsTree(IVsProfileSettingsTree settingsTree)

{

logIndent(1); scope(exit) logIndent(-1);

BSTR bname;

string name, desc, cat, regname, nameForId, fullPath, pkg;

if(SUCCEEDED(settingsTree.GetDisplayName(&bname)))

name = detachBSTR(bname);

if(SUCCEEDED(settingsTree.GetDescription(&bname)))

desc = detachBSTR(bname);

if(SUCCEEDED(settingsTree.GetCategory(&bname)))

cat = detachBSTR(bname);

if(SUCCEEDED(settingsTree.GetRegisteredName(&bname)))

regname = detachBSTR(bname);

if(SUCCEEDED(settingsTree.GetNameForID(&bname)))

nameForId = detachBSTR(bname);

if(SUCCEEDED(settingsTree.GetFullPath(&bname)))

fullPath = detachBSTR(bname);

if(SUCCEEDED(settingsTree.GetPackage(&bname)))

pkg = detachBSTR(bname);

logCall("Name: " ~ name ~ ", Desc: " ~ desc ~ ", Cat: " ~ cat ~ ", regname: " ~ regname, ", nameForId: " ~ nameForId ~ ", fullpath: " ~ fullPath ~ ", pkg: " ~ pkg);

int count;

if(SUCCEEDED(settingsTree.GetChildCount(&count)))

for(int i = 0; i < count; i++)

{

IVsProfileSettingsTree child;

if(SUCCEEDED(settingsTree.GetChild(i, &child)))

{

scope(exit) release(child);

logSettingsTree(child);

}

}

}

version(none)

string[] getVCLibraryPaths()

{

IVsProfileDataManager pdm = queryService!(SVsProfileDataManager,IVsProfileDataManager)();

if(!pdm)

return null;

scope(exit) release(pdm);

IVsProfileSettingsTree settingsTree;

HRESULT hr = pdm.GetSettingsForExport(&settingsTree);

if(SUCCEEDED(hr))

{

scope(exit) release(settingsTree);

logSettingsTree(settingsTree);

}

//pdm.ShowProfilesUI();

return null;

}

bool buildPhobosBrowseInfo()

{

IVsOutputWindowPane pane = getVisualDOutputPane();

if(!pane)

return false;

scope(exit) release(pane);

string[] jsonPaths = getJSONPaths();

string jsonPath;

if(jsonPaths.length)

jsonPath = jsonPaths[0];

if(jsonPath.length == 0)

{

JSNSearchPath ~= "\"$(APPDATA)[\\VisualD\\json\\\](file:///\\VisualD\json\)"";

saveToRegistry();

jsonPath = getJSONPaths()[0];

}

pane.Clear();

pane.Activate();

string msg = "Building phobos JSON browse information files to " ~ jsonPath ~ "\n";

pane.OutputString(toUTF16z(msg));

if(!std.file.exists(jsonPath))

{

try

{

mkdirRecurse(jsonPath[0..$-1]); // normalized dir has trailing slash

}

catch(Exception)

{

return OutputErrorString(msg = "cannot create directory " ~ jsonPath);

}

}

string[] imports = getIniImportPaths();

foreach(s; imports)

pane.OutputString(toUTF16z("Using import " ~ s ~ "\n"));

string cmdfile = jsonPath ~ "buildjson.bat";

string dmddir = findDmdBinDir();

string dmdpath = dmddir ~ "dmd.exe";

if(!std.file.exists(dmdpath))

return OutputErrorString(msg = "dmd.exe not found in DMDInstallDir=" ~ DMD.InstallDir ~ " or through PATH");

foreach(s; imports)

{

string[] files;

string cmdline = "@echo off\n";

string jsonfile;

string opts = " -d -c -o-";

if(std.file.exists(s ~ "std\\algorithm.d") || std.file.exists(s ~ "std\\algorithm\\package.d")) // D2

{

files ~= findDRuntimeFiles(s, "std", true);

files ~= findDRuntimeFiles(s, "etc\\c", true);

jsonfile = jsonPath ~ "phobos.json";

}

if(std.file.exists(s ~ "std\\gc.d")) // D1

{

files ~= findDRuntimeFiles(s, "std", false);

files ~= findDRuntimeFiles(s, "std\\c", false);

files ~= findDRuntimeFiles(s, "std\\c\\windows", false);

files ~= findDRuntimeFiles(s, "std\\windows", false);

jsonfile = jsonPath ~ "phobos1.json";

}

if(std.file.exists(s ~ "object.di") || std.file.exists(s ~ "object.d"))

{

opts ~= " -I" ~ buildPath(s, "..[\\src](file:///\\src)"); // needed since dmd 2.059

if (std.file.exists(s ~ "object.di"))

files ~= "object.di";

else

files ~= "object.d"; // dmd >=2.068 no longer has object.di

files ~= findDRuntimeFiles(s, "core", true);

files ~= findDRuntimeFiles(s, "std", false); // D1?

jsonfile = jsonPath ~ "druntime.json";

}

if(files.length)

{

string sfiles = std.string.join(files, " ");

cmdline ~= quoteFilename(dmdpath) ~ opts ~ " -Xf" ~ quoteFilename(jsonfile) ~ " " ~ sfiles ~ "\n\n";

pane.OutputString(toUTF16z("Building " ~ jsonfile ~ " from import " ~ s ~ "\n"));

if(!launchBuildPhobos(s, cmdfile, cmdline, pane))

pane.OutputString(toUTF16z("Building " ~ jsonfile ~ " failed!\n"));

else

pane.OutputString(toUTF16z("Building " ~ jsonfile ~ " successful!\n"));

}

}

return true;

}

string findCoverageFile(string srcfile)

{

import stdext.path;

import std.path;

import std.file;

import std.string;

string lstname = std.path.stripExtension(srcfile) ~ ".lst";

if(std.file.exists(lstname) && std.file.isFile(lstname))

return lstname;

string srcpath = stripExtension(toLower(makeFilenameCanonical(srcfile, "")));

foreach(dir; coverageExecutionDirs)

{

if(!std.file.exists(dir) || !std.file.isDir(dir))

continue;

foreach(string f; dirEntries(dir, SpanMode.shallow))

{

char[] fn = baseName(f).dup;

toLowerInPlace(fn);

auto ext = extension(fn);

if(ext != ".lst")

continue;

// assume no '-' in file name, cov replaced '\\' with these

bool isAbs = false;

if(std.ascii.isAlpha(fn[0]) && fn[1] == '-' && fn[2] == '-')

{

// absolute path

fn[1] = ':';

isAbs = true;

}

for(size\_t i = 0; i < fn.length; i++)

if(fn[i] == '-')

fn[i] = '\\';

string fs = to!string(fn);

if(isAbs)

{

fs = removeDotDotPath(fs);

if(fs[0 .. $-4] == srcpath)

return std.path.buildPath(dir, f);

}

else

{

foreach(bdir; coverageBuildDirs)

{

string bfile = toLower(makeFilenameCanonical(fs, bdir));

if(bfile[0 .. $-4] == srcpath)

return std.path.buildPath(dir, f);

}

}

}

}

return null;

}

void DeleteCoverageFiles()

{

string[] dirs = coverageExecutionDirs;

Source[] srcs = Package.GetLanguageService().GetSources();

foreach(src; srcs)

dirs.addunique(dirName(src.GetFileName()));

foreach(dir; dirs)

if(std.file.exists(dir) && std.file.isDir(dir))

{

string[] lstfiles;

foreach(f; std.file.dirEntries(dir, SpanMode.shallow))

if(icmp(extension(f.name), ".lst") == 0)

lstfiles ~= f;

foreach(lst; lstfiles)

collectException(std.file.remove(lst));

}

coverageExecutionDirs = null;

}

void addExecutionPath(string dir, string workdir = null)

{

dir = makeDirnameCanonical(dir, workdir);

adduniqueLRU(coverageExecutionDirs, dir, 4);

}

void addBuildPath(string dir, string workdir = null)

{

dir = makeDirnameCanonical(dir, workdir);

adduniqueLRU(coverageBuildDirs, dir, 4);

}

//////////////////////////////////////////////////////////////////////////////

void updateDefaultColors()

{

scope RegKey keyUserOpts = new RegKey(hUserKey, regConfigRoot ~ regPathToolsOptions, false);

bool wasDark = keyUserOpts.GetDWORD("lastThemeWasDark") != 0;

bool isDark = isDarkTheme();

if (wasDark != isDark)

{

scope RegKey keyUserOptsWr = new RegKey(hUserKey, regConfigRoot ~ regPathToolsOptions, true);

removeColorCache();

keyUserOptsWr.Set("lastThemeWasDark", isDark);

}

LanguageService.updateThemeColors();

}

bool isDarkTheme()

{

scope RegKey keyUserOpts = new RegKey(hUserKey, regUserRoot ~ r"\General", false);

string theme = toUTF8(keyUserOpts.GetString("CurrentTheme")).toLower;

if (theme == "1ded0138-47ce-435e-84ef-9ec1f439b749" || theme == "{1ded0138-47ce-435e-84ef-9ec1f439b749}")

return true;

// VS2015

scope RegKey keyUserOpts15 = new RegKey(hUserKey, regUserRoot ~ r"\ApplicationPrivateSettings\Microsoft\VisualStudio", false);

string theme15 = toUTF8(keyUserOpts15.GetString("ColorTheme")).toLower;

return theme15.endsWith("1ded0138-47ce-435e-84ef-9ec1f439b749");

}

bool removeColorCache()

{

auto hr = RegDeleteRecursive(hUserKey, regUserRoot ~ r"\FontAndColors\Cache\{E0187991-B458-4F7E-8CA9-42C9A573B56C}");

return SUCCEEDED(hr);

}

}

///////////////////////////////////////////////////////////////////////

class WizardFactory : DComObject, IClassFactory

{

override HRESULT QueryInterface(in IID\* riid, void\*\* pvObject)

{

if(queryInterface2!(IClassFactory) (this, IID\_IClassFactory, riid, pvObject))

return S\_OK;

return super.QueryInterface(riid, pvObject);

}

override HRESULT CreateInstance(IUnknown UnkOuter, in IID\* riid, void\*\* pvObject)

{

logCall("%s.CreateInstance(riid=%s)", this, \_toLog(riid));

assert(!UnkOuter);

auto wiz = newCom!ItemWizard;

return wiz.QueryInterface(riid, pvObject);

}

override HRESULT LockServer(in BOOL fLock)

{

if(fLock)

InterlockedIncrement(&g\_dllRefCount);

else

InterlockedDecrement(&g\_dllRefCount);

return S\_OK;

}

int lockCount;

}

class ItemWizard : DisposingDispatchObject, dte.IDTWizard

{

override HRESULT QueryInterface(in IID\* riid, void\*\* pvObject)

{

if(queryInterface!(dte.IDTWizard) (this, riid, pvObject))

return S\_OK;

return super.QueryInterface(riid, pvObject);

}

override void Dispose()

{

}

override ComTypeInfoHolder getTypeHolder ()

{

mixin(LogCallMix);

return null;

}

override HRESULT Execute(/+[in]+/ IDispatch Application,

in int hwndOwner,

in SAFEARRAY\* ContextParams,

in SAFEARRAY\* CustomParams,

/+[in, out]+/ dte.wizardResult\* retval)

{

mixin(LogCallMix);

SAFEARRAY\* sa = \*cast(SAFEARRAY\*\*)ContextParams;

assert(SafeArrayGetDim(sa) == 1);

LONG lbound, ubound;

SafeArrayGetLBound(sa, 1, &lbound);

SafeArrayGetUBound(sa, 1, &ubound);

size\_t cnt = (ubound - lbound + 1);

string WizardType, ProjectName, /\*ProjectItems,\*/ LocalDirectory, ItemName, InstallationDirectory;

bool silent;

VARTYPE vt;

SafeArrayGetVartype(sa, &vt);

if(vt == VT\_VARIANT)

{

VARIANT var;

LONG idx = lbound;

if(SafeArrayGetElement(sa, &idx, &var) == S\_OK && var.vt == VT\_BSTR)

WizardType = to\_string(var.bstrVal);

if(SafeArrayGetElement(sa, &++idx, &var) == S\_OK && var.vt == VT\_BSTR)

ProjectName = to\_string(var.bstrVal);

++idx;

if(SafeArrayGetElement(sa, &++idx, &var) == S\_OK && var.vt == VT\_BSTR)

LocalDirectory = to\_string(var.bstrVal);

if(SafeArrayGetElement(sa, &++idx, &var) == S\_OK && var.vt == VT\_BSTR)

ItemName = to\_string(var.bstrVal);

if(SafeArrayGetElement(sa, &++idx, &var) == S\_OK && var.vt == VT\_BSTR)

InstallationDirectory = to\_string(var.bstrVal);

if(SafeArrayGetElement(sa, &++idx, &var) == S\_OK && var.vt == VT\_BOOL)

silent = var.boolVal != 0;

}

UtilMessageBox("Sorry, it does not make sense to add a package without specifying a folder.\n"

"Please use the \"Add new item\" command from the project context menu.",

MB\_OK, "Visual D - Add package");

if(retval)

\*retval = dte.wizardResultCancel;

return S\_OK;

}

}

// This file is part of Visual D

//

// Visual D integrates the D programming language into Visual Studio

// Copyright (c) 2010 by Rainer Schuetze, All Rights Reserved

//

// Distributed under the Boost Software License, Version 1.0.

// See accompanying file LICENSE\_1\_0.txt or copy at <http://www.boost.org/LICENSE_1_0.txt>

module visuald.dproject;

import visuald.windows;

import core.stdc.string : memcpy;

import core.stdc.wchar\_ : wcslen;

import core.thread;

import std.windows.charset;

import std.string;

import std.utf;

import std.file;

import std.path;

import std.conv;

import std.array;

import stdext.path;

import xml = visuald.xmlwrap;

import sdk.win32.rpcdce;

import sdk.win32.oleauto;

import sdk.win32.objbase;

import sdk.vsi.vsshell;

import sdk.vsi.vsshell80;

import sdk.vsi.vsshell90;

import sdk.vsi.ivssccproject2;

import sdk.vsi.fpstfmt;

import dte = sdk.vsi.dte80a;

import visuald.comutil;

import visuald.logutil;

import visuald.automation;

import visuald.dpackage;

import visuald.propertypage;

import visuald.hierarchy;

import visuald.hierutil;

import visuald.fileutil;

import visuald.chiernode;

import visuald.chiercontainer;

import visuald.build;

import visuald.config;

import visuald.oledatasource;

import visuald.pkgutil;

import visuald.dimagelist;

import visuald.dllmain : g\_hInst;

///////////////////////////////////////////////////////////////

class ProjectFactory : DComObject, IVsProjectFactory

{

this(Package pkg)

{

//mPackage = pkg;

}

override HRESULT QueryInterface(in IID\* riid, void\*\* pvObject)

{

//mixin(LogCallMix);

if(queryInterface!(IVsProjectFactory) (this, riid, pvObject))

return S\_OK;

return super.QueryInterface(riid, pvObject);

}

override int CanCreateProject(in wchar\* pszFilename, in uint grfCreateFlags, int\* pfCanCreate)

{

mixin(LogCallMix);

\*pfCanCreate = 1;

return S\_OK;

}

override int Close()

{

mixin(LogCallMix);

return S\_OK;

}

override int CreateProject(in wchar\* pszFilename, in wchar\* pszLocation, in wchar\* pszName, in VSCREATEPROJFLAGS grfCreateFlags,

in IID\* iidProject, void\*\* ppvProject, BOOL\* pfCanceled)

{

mixin(LogCallMix);

version(none)

{

CoInitialize(null);

                        VCProjectEngine spEngine;

int hr = CoCreateInstance(&VCProjectEngineObject.iid, null, CLSCTX\_INPROC\_SERVER, &VCProjectEngine.iid, cast(void\*)&spEngine);

if( hr != S\_OK || !spEngine )

{

CoUninitialize();

return returnError(E\_FAIL);

}

// Open an existing project.

IDispatch \*spDispProj = spEngine.CreateProject(pszFilename);

if(!spDispProj)

{

CoUninitialize();

return returnError(E\_FAIL);

}

} // version

if(grfCreateFlags & CPF\_OPENFILE)

{

string filename = to\_string(pszFilename);

string name = baseName(filename);

Project prj = newCom!Project(this, name, filename);

\*pfCanceled = 0;

return prj.QueryInterface(iidProject, ppvProject);

}

else if(grfCreateFlags & CPF\_CLONEFILE)

{

string src = to\_string(pszFilename);

string name = to\_string(pszName);

string dest = to\_string(pszLocation) ~ name ~ "." ~ toUTF8(g\_projectFileExtensions);

if(!cloneProject(src, dest))

return returnError(E\_FAIL);

//std.file.copy(to\_wstring(pszFilename), to\_wstring(pszLocation));

Project prj = newCom!Project(this, name, dest);

\*pfCanceled = 0;

return prj.QueryInterface(iidProject, ppvProject);

}

return returnError(E\_NOTIMPL);

}

override int SetSite(IServiceProvider psp)

{

mixin(LogCallMix);

return returnError(E\_NOTIMPL);

}

///////////////////////////////////////////////////////////////

bool cloneProjectFiles(string srcdir, string destdir, xml.Element node)

{

xml.Element[] folderItems = xml.elementsById(node, "Folder");

foreach(folder; folderItems)

if (!cloneProjectFiles(srcdir, destdir, folder))

return false;

xml.Element[] fileItems = xml.elementsById(node, "File");

foreach(file; fileItems)

{

string fileName = xml.getAttribute(file, "path");

std.file.copy(srcdir ~ fileName, destdir ~ fileName);

}

return true;

}

bool cloneProject(string src, string dest)

{

try

{

string srcdir = dirName(src) ~ "\\";

string destdir = dirName(dest) ~ "\\";

auto doc = Project.readXML(src);

if(!doc)

return false;

if(!cloneProjectFiles(srcdir, destdir, xml.getRoot(doc)))

return false;

if(!Project.saveXML(doc, dest))

return false;

return true;

}

catch(Exception e)

{

writeToBuildOutputPane(e.msg);

logCall(e.toString());

}

return false;

}

private:

//Package mPackage;

}

///////////////////////////////////////////////////////////////////////

class Project : CVsHierarchy,

IVsProject,

IVsParentProject,

IVsGetCfgProvider,

IVsProject3,

IVsHierarchyDeleteHandler,

IVsAggregatableProject,

IVsProjectFlavorCfgProvider,

IPersistFileFormat,

IVsProjectBuildSystem,

IVsBuildPropertyStorage,

IVsComponentUser,

IVsDependencyProvider,

ISpecifyPropertyPages,

IPerPropertyBrowsing,

dte.IVsGlobalsCallback,

IVsHierarchyDropDataSource2,

IVsHierarchyDropDataTarget,

IVsNonLocalProject,

//IRpcOptions,

IVsSccProject2,

//IBuildDependencyUpdate,

//IProjectEventsListener,

//IProjectEventsProvider,

//IReferenceContainerProvider,

IVsProjectSpecialFiles

{

static const GUID iid = { 0x5840c881, 0x9d9e, 0x4a85, [ 0xb7, 0x6b, 0x50, 0xa9, 0x68, 0xdb, 0x22, 0xf9 ] };

this(ProjectFactory factory, string name, string filename)

{

mFactory = factory;

mCaption = mName = name;

mFilename = filename;

mConfigProvider = addref(newCom!ConfigProvider(this));

parseXML();

}

this(ProjectFactory factory, string name, string filename, string platform, string config)

{

mFactory = factory;

mCaption = mName = name;

mFilename = filename;

mConfigProvider = addref(newCom!ConfigProvider(this));

mConfigProvider.addConfig(platform, config);

CProjectNode rootnode = newCom!CProjectNode(filename, this);

rootnode.SetName(name);

SetRootNode(rootnode);

}

override void Dispose()

{

mConfigProvider = release(mConfigProvider);

//mExtProject = release(mExtProject);

super.Dispose();

}

override HRESULT QueryInterface(in IID\* riid, void\*\* pvObject)

{

//mixin(LogCallMix);

if(queryInterface!(Project) (this, riid, pvObject))

return S\_OK;

if(queryInterface!(IVsProject) (this, riid, pvObject))

return S\_OK;

if(queryInterface!(IVsProject2) (this, riid, pvObject))

return S\_OK;

if(queryInterface!(IVsProject3) (this, riid, pvObject))

return S\_OK;

if(queryInterface!(IVsHierarchyDeleteHandler) (this, riid, pvObject))

return S\_OK;

//                if(queryInterface!(IVsParentProject) (this, riid, pvObject))

//                        return S\_OK;

if(queryInterface!(IVsGetCfgProvider) (this, riid, pvObject))

return S\_OK;

if(queryInterface!(ISpecifyPropertyPages) (this, riid, pvObject))

return S\_OK;

if(queryInterface!(IVsAggregatableProject) (this, riid, pvObject))

return S\_OK;

if(queryInterface!(IVsProjectFlavorCfgProvider) (this, riid, pvObject))

return S\_OK;

if(queryInterface!(IPersist) (this, riid, pvObject))

return S\_OK;

if(queryInterface!(IPersistFileFormat) (this, riid, pvObject))

return S\_OK;

if(queryInterface!(IVsProjectBuildSystem) (this, riid, pvObject))

return S\_OK;

if(queryInterface!(IVsBuildPropertyStorage) (this, riid, pvObject))

return S\_OK;

if(queryInterface!(IVsComponentUser) (this, riid, pvObject))

return S\_OK;

//if(queryInterface!(IVsDependencyProvider) (this, riid, pvObject))

//        return S\_OK;

if(queryInterface!(dte.IVsGlobalsCallback) (this, riid, pvObject))

return S\_OK;

if(queryInterface!(IVsHierarchyDropDataSource) (this, riid, pvObject))

return S\_OK;

if(queryInterface!(IVsHierarchyDropDataSource2) (this, riid, pvObject))

return S\_OK;

if(queryInterface!(IVsHierarchyDropDataTarget) (this, riid, pvObject))

return S\_OK;

if(queryInterface!(IVsNonLocalProject) (this, riid, pvObject))

return S\_OK;

if(queryInterface!(IVsSccProject2) (this, riid, pvObject))

return S\_OK;

//if(queryInterface!(IRpcOptions) (this, riid, pvObject))

//        return S\_OK;

//if(queryInterface!(IPerPropertyBrowsing) (this, riid, pvObject))

//        return S\_OK;

return super.QueryInterface(riid, pvObject);

}

// IDispatch

\_\_gshared ComTypeInfoHolder mTypeHolder;

static void shared\_static\_this\_typeHolder()

{

static class \_ComTypeInfoHolder : ComTypeInfoHolder

{

override int GetIDsOfNames(

/\* [size\_is][in] \*/ in LPOLESTR \*rgszNames,

/\* [in] \*/ in UINT cNames,

/\* [size\_is][out] \*/ MEMBERID \*pMemId)

{

//mixin(LogCallMix);

if (cNames == 1 && to\_string(\*rgszNames) == "Name")

{

\*pMemId = 1;

return S\_OK;

}

if (cNames == 1 && to\_string(\*rgszNames) == "\_\_id")

{

\*pMemId = 2;

return S\_OK;

}

return returnError(E\_NOTIMPL);

}

}

mTypeHolder = newCom!\_ComTypeInfoHolder;

addref(mTypeHolder);

}

static void shared\_static\_dtor\_typeHolder()

{

mTypeHolder = release(mTypeHolder);

}

override ComTypeInfoHolder getTypeHolder () { return mTypeHolder; }

override int Invoke(

/\* [in] \*/ in DISPID dispIdMember,

/\* [in] \*/ in IID\* riid,

/\* [in] \*/ in LCID lcid,

/\* [in] \*/ in WORD wFlags,

/\* [out][in] \*/ DISPPARAMS \*pDispParams,

/\* [out] \*/ VARIANT \*pVarResult,

/\* [out] \*/ EXCEPINFO \*pExcepInfo,

/\* [out] \*/ UINT \*puArgErr)

{

mixin(LogCallMix);

if(dispIdMember == 1 || dispIdMember == 2)

{

if(pDispParams.cArgs == 0)

return GetProperty(VSITEMID\_ROOT, VSHPROPID\_Name, pVarResult);

}

return returnError(E\_NOTIMPL);

}

// IVsProject

override int IsDocumentInProject(in LPCOLESTR pszMkDocument, BOOL\* pfFound, VSDOCUMENTPRIORITY\* pdwPriority, VSITEMID\* pitemid)

{

mixin(LogCallMix);

string docName = to\_string(pszMkDocument);

if(!isAbsolute(docName))

{

string root = dirName(GetRootNode().GetFullPath());

docName = root ~ "\\" ~ docName;

}

docName = toLower(docName);

CHierNode node = searchNode(GetRootNode(), delegate (CHierNode n) { return n.GetCanonicalName() == docName; });

if(node)

{

if(pfFound) \*pfFound = true;

if(pitemid) \*pitemid = node is GetRootNode() ? VSITEMID\_ROOT : node.GetVsItemID();

if (pdwPriority) \*pdwPriority = cast(CFileNode) node ? DP\_Standard : DP\_Intrinsic;

}

else

{

if(pfFound) \*pfFound = false;

if(pitemid) \*pitemid = VSITEMID\_NIL;

if (pdwPriority) \*pdwPriority = DP\_Unsupported;

}

return S\_OK;

}

override int OpenItem(in VSITEMID itemid, in GUID\* rguidLogicalView, IUnknown punkDocDataExisting, IVsWindowFrame \*ppWindowFrame)

{

mixin(LogCallMix);

if(CFileNode pNode = cast(CFileNode) VSITEMID2Node(itemid))

return OpenDoc(pNode, false /\*fNewFile\*/,

false /\*fUseOpenWith\*/,

false /\*fShow\*/,

rguidLogicalView,

&GUID\_NULL, null,

punkDocDataExisting,

ppWindowFrame);

return returnError(E\_UNEXPECTED);

}

override int GetItemContext(in VSITEMID itemid, IServiceProvider\* ppSP)

{

logCall("GetItemContext(itemid=%s, ppSP=%s)", \_toLog(itemid), \_toLog(ppSP));

// NOTE: this method allows a project to provide project context services

// to an item (document) editor. If the project does not need to provide special

// services to its items then it should return null. Under no circumstances

// should you return the IServiceProvider pointer that was passed to our

// package from the Environment via IVsPackage::SetSite. The global services

// will automatically be made available to editors.

\*ppSP = null;

return S\_OK;

}

override int GenerateUniqueItemName(in VSITEMID itemidLoc, in wchar\* pszExt, in wchar\* pszSuggestedRoot, BSTR \*pbstrItemName)

{

mixin(LogCallMix);

// as we are using virtual folders, just suggest a file in the project directory

string dir = dirName(GetProjectNode().GetFullPath());

string root = pszSuggestedRoot ? to\_string(pszSuggestedRoot) : "File";

string ext = pszExt ? to\_string(pszExt) : ".d";

for(int i = 1; i < int.max; i++)

{

string file = dir ~ "\\" ~ root ~ format("%d", i) ~ ext;

if(!std.file.exists(file))

{

\*pbstrItemName = allocBSTR(file);

return S\_OK;

}

}

return returnError(E\_FAIL);

}

override int GetMkDocument(in VSITEMID itemid, BSTR \*pbstrMkDocument)

{

mixin(LogCallMix2);

//logCall("%s.GetMkDocument(this=%s, itemid=%s, pbstrMkDocument=%s)", this, cast(void\*)this, \_toLog(itemid), \_toLog(pbstrMkDocument));

if(CHierNode pNode = VSITEMID2Node(itemid))

{

\*pbstrMkDocument = allocBSTR(pNode.GetFullPath());

logCall("%s.GetMkDocument returns pbstrMkDocument=%s", this, to\_string(\*pbstrMkDocument));

return S\_OK;

}

return returnError(E\_INVALIDARG);

}

override int AddItem(in VSITEMID itemidLoc, in VSADDITEMOPERATION dwAddItemOperation,

in LPCOLESTR pszItemName,

in ULONG cFilesToOpen, in LPCOLESTR \* rgpszFilesToOpen,

in HWND hwndDlgOwner, VSADDRESULT\* pResult)

{

mixin(LogCallMix);

return AddItemWithSpecific(

/\* [in] VSITEMID itemidLoc \*/ itemidLoc,

/\* [in] VSADDITEMOPERATION dwAddItemOperation \*/ dwAddItemOperation,

/\* [in] LPCOLESTR pszItemName \*/ pszItemName,

/\* [in] ULONG cFilesToOpen \*/ cFilesToOpen,

/\* [in] LPCOLESTR rgpszFilesToOpen[] \*/ rgpszFilesToOpen,

/\* [in] HWND hwndDlg \*/ hwndDlgOwner,

/\* [in] VSSPECIFICEDITORFLAGS grfEditorFlags \*/ VSSPECIFICEDITOR\_DoOpen | VSSPECIFICEDITOR\_UseView,

/\* [in] REFGUID rguidEditorType \*/ &GUID\_NULL,

/\* [in] LPCOLESTR pszPhysicalView \*/ null,

/\* [in] REFGUID rguidLogicalView \*/ &GUID\_NULL, //LOGVIEWID\_Primary,

/\* [out] VSADDRESULT \* pResult \*/ pResult);

}

// IVsProject2

override int RemoveItem(

/\* [in] \*/ in DWORD dwReserved,

/\* [in] \*/ in VSITEMID itemid,

/\* [retval][out] \*/ BOOL \*pfResult)

{

mixin(LogCallMix);

if(itemid == VSITEMID\_ROOT || itemid == VSITEMID\_NIL)

return E\_UNEXPECTED;

int hr = DeleteItem(DELITEMOP\_RemoveFromProject, itemid);

\*pfResult = SUCCEEDED(hr);

return hr;

}

override int ReopenItem(

/\* [in] \*/ in VSITEMID itemid,

/\* [in] \*/ in GUID\* rguidEditorType,

/\* [in] \*/ in wchar\* pszPhysicalView,

/\* [in] \*/ in GUID\* rguidLogicalView,

/\* [in] \*/ IUnknown punkDocDataExisting,

/\* [retval][out] \*/ IVsWindowFrame \*ppWindowFrame)

{

mixin(LogCallMix);

if(CFileNode pNode = cast(CFileNode) VSITEMID2Node(itemid))

return OpenDoc(pNode, false /\*fNewFile\*/,

false /\*fUseOpenWith\*/,

false /\*fShow\*/,

rguidLogicalView,

rguidEditorType, pszPhysicalView,

punkDocDataExisting,

ppWindowFrame);

return returnError(E\_UNEXPECTED);

}

// IVsProject3

override int AddItemWithSpecific(

/\* [in] \*/ in VSITEMID itemidLoc,

/\* [in] \*/ in VSADDITEMOPERATION dwAddItemOperation,

/\* [in] \*/ in wchar\* pszItemName,

/\* [in] \*/ in uint cFilesToOpen,

/\* [size\_is][in] \*/ in LPCOLESTR\* rgpszFilesToOpen,

/\* [in] \*/ in HWND hwndDlgOwner,

/\* [in] \*/ in VSSPECIFICEDITORFLAGS grfEditorFlags,

/\* [in] \*/ in GUID\* rguidEditorType,

/\* [in] \*/ in LPCOLESTR pszPhysicalView,

/\* [in] \*/ in GUID\* rguidLogicalView,

/\* [retval][out] \*/ VSADDRESULT\* pResult)

{

// AddItemWithSpecific is used to add item(s) to the project and

// additionally ask the project to open the item using the specified

// editor information. An extension of IVsProject::AddItem().

mixin(LogCallMix);

if(CHierContainer pNode = cast(CHierContainer) VSITEMID2Node(itemidLoc))

{

return AddItemSpecific(pNode,

/\* [in] VSADDITEMOPERATION dwAddItemOperation \*/ dwAddItemOperation,

/\* [in] LPCOLESTR pszItemName \*/ pszItemName,

/\* [in] DWORD cFilesToOpen \*/ cFilesToOpen,

/\* [in] LPCOLESTR rgpszFilesToOpen[] \*/ rgpszFilesToOpen,

/\* [in] HWND hwndDlg \*/ hwndDlgOwner,

/\* [in] VSSPECIFICEDITORFLAGS grfEditorFlags \*/ grfEditorFlags,

/\* [in] REFGUID rguidEditorType \*/ rguidEditorType,

/\* [in] LPCOLESTR pszPhysicalView \*/ pszPhysicalView,

/\* [in] REFGUID rguidLogicalView\*/ rguidLogicalView,

/\* [in] bool moveIfInProject \*/ false,

/\* [out] VSADDRESULT \*pResult \*/ pResult);

}

return returnError(E\_UNEXPECTED);

}

override int OpenItemWithSpecific(

/\* [in] \*/ in VSITEMID itemid,

/\* [in] \*/ in VSSPECIFICEDITORFLAGS grfEditorFlags,

/\* [in] \*/ in GUID\* rguidEditorType,

/\* [in] \*/ in wchar\* pszPhysicalView,

/\* [in] \*/ in GUID\* rguidLogicalView,

/\* [in] \*/ IUnknown punkDocDataExisting,

/\* [out] \*/ IVsWindowFrame \*ppWindowFrame)

{

mixin(LogCallMix);

return returnError(E\_NOTIMPL);

}

override int TransferItem(

/\* [in] \*/ in wchar\* pszMkDocumentOld,

/\* [in] \*/ in wchar\* pszMkDocumentNew,

/\* [in] \*/ IVsWindowFrame punkWindowFrame)

{

mixin(LogCallMix);

return returnError(E\_NOTIMPL);

}

override int QueryDeleteItem(

/\* [in] \*/ in VSDELETEITEMOPERATION dwDelItemOp,

/\* [in] \*/ in VSITEMID itemid,

/\* [retval][out] \*/ BOOL \*pfCanDelete)

{

//                mixin(LogCallMix);

bool canDelete = true;

if(dwDelItemOp == DELITEMOP\_DeleteFromStorage)

{

CHierNode[] nodes = VSITEMID2Nodes(itemid);

foreach(n; nodes)

if(cast(CHierContainer) n)

canDelete = false;

}

\*pfCanDelete = canDelete; //(dwDelItemOp == DELITEMOP\_RemoveFromProject);

return S\_OK;

}

override int DeleteItem(

/\* [in] \*/ in VSDELETEITEMOPERATION dwDelItemOp,

/\* [in] \*/ in VSITEMID itemid)

{

mixin(LogCallMix);

// the root item will be removed without asking the project itself

if(itemid == VSITEMID\_ROOT || itemid == VSITEMID\_NIL) // || dwDelItemOp != DELITEMOP\_RemoveFromProject)

return E\_INVALIDARG;

CHierNode[] nodes = VSITEMID2Nodes(itemid);

if(nodes.length == 0)

return S\_OK;

version(none)

{

int delFiles = Package.GetGlobalOptions().deleteFiles;

if(delFiles == 0)

{

string sfiles = (nodes.length == 1 ? "file" : to!string(nodes.length) ~ " files");

int answer = UtilMessageBox("Do you want to delete the " ~ sfiles ~ " on disk?\n\n" ~

"You can permanently answer this dialog in the global Visual D settings.", MB\_YESNOCANCEL | MB\_ICONEXCLAMATION,

"Remove file from project");

if(answer == IDCANCEL)

return S\_FALSE;

if(answer == IDYES)

delFiles = 1;

else

delFiles = -1;

}

}

foreach(node; nodes)

{

if(!node)

return E\_INVALIDARG;

if(CFileNode fnode = cast(CFileNode) node)

{

string fname = fnode.GetFullPath();

if(HRESULT hr = fnode.CloseDoc(SLNSAVEOPT\_PromptSave))

return hr;

if(dwDelItemOp == DELITEMOP\_DeleteFromStorage)

moveFileToRecycleBin(fname);

//std.file.remove(fname);

}

if(node.GetParent()) // might be already removed because folder has been removed?

node.GetParent().Delete(node, this);

}

return S\_OK;

}

// IVsHierarchy

override int Close()

{

mixin(LogCallMix);

if(int rc = super.Close())

return rc;

return S\_OK;

}

override int GetGuidProperty(in VSITEMID itemid, in VSHPROPID propid, GUID\* pguid)

{

mixin(LogCallMix);

if(itemid == VSITEMID\_ROOT)

{

switch(propid)

{

case VSHPROPID\_ProjectIDGuid:

\*pguid = mProjectGUID;

return S\_OK;

case VSHPROPID\_TypeGuid:

\*pguid = g\_projectFactoryCLSID;

return S\_OK;

default:

break;

}

}

return super.GetGuidProperty(itemid, propid, pguid);

}

/\*override\*/ int SetGuidProperty(in VSITEMID itemid, in VSHPROPID propid, in GUID\* rguid)

{

mixin(LogCallMix2);

if(propid != VSHPROPID\_ProjectIDGuid)

return returnError(E\_NOTIMPL);

if(itemid != VSITEMID\_ROOT)

return returnError(E\_INVALIDARG);

mProjectGUID = \*rguid;

return S\_OK;

}

override int GetProperty(in VSITEMID itemid, in VSHPROPID propid, VARIANT\* var)

{

//mixin(LogCallMix);

if(itemid == VSITEMID\_ROOT)

{

// handle project specific stuff before generic node properties

switch(propid)

{

case VSHPROPID\_ExtObject:

var.vt = VT\_DISPATCH;

if(!mExtProject)

mExtProject = /\*addref\*/(newCom!ExtProject(this));

var.pdispVal = addref(mExtProject);

return S\_OK;

default:

break;

}

}

if(super.GetProperty(itemid, propid, var) == S\_OK)

return S\_OK;

if(itemid != VSITEMID\_ROOT)

{

logCall("Getting unknown property %d for item %x!", propid, itemid);

return returnError(DISP\_E\_MEMBERNOTFOUND);

}

switch(propid)

{

case VSHPROPID\_TypeName:

var.vt = VT\_BSTR;

var.bstrVal = allocBSTR("typename");

break;

case VSHPROPID\_SaveName:

var.vt = VT\_BSTR;

var.bstrVal = allocBSTR(mFilename);

break;

version(none)

{

case VSHPROPID\_ProductBrandName:

var.vt = VT\_BSTR;

var.bstrVal = allocBSTR("VisualD");

break;

}

case VSHPROPID\_BrowseObject:

var.vt = VT\_DISPATCH;

return QueryInterface(&IDispatch.iid, cast(void \*\*)&var.pdispVal);

case VSHPROPID\_ConfigurationProvider:

var.vt = VT\_UNKNOWN;

return GetCfgProvider(cast(IVsCfgProvider\*)&var.punkVal);

//return QueryInterface(&IVsGetCfgProvider.iid, cast(void \*\*)&var.punkVal);

case VSHPROPID\_ProjectDir:

// IsNonSearchable, HasEnumerationSideEffects

// 1001

//case VSHPROPID2.EnableDataSourceWindow:

//case VSHPROPID2.DebuggeeProcessId:

case cast(VSHPROPID) 1001:

default:

logCall("Getting unknown property %d for item %x!", propid, itemid);

return DISP\_E\_MEMBERNOTFOUND;

// return returnError(E\_NOTIMPL); // DISP\_E\_MEMBERNOTFOUND;

}

return S\_OK;

}

override int SetProperty(in VSITEMID itemid, in VSHPROPID propid, in VARIANT var)

{

mixin(LogCallMix);

switch(propid)

{

case VSHPROPID\_Caption:

if(var.vt != VT\_BSTR)

return returnError(E\_INVALIDARG);

mCaption = to\_string(var.bstrVal);

break;

default:

HRESULT hr = super.SetProperty(itemid, propid, var);

if(hr == S\_OK)

break;

logCall("Setting unknown property %d on %x!", propid, itemid);

return hr;

}

return S\_OK;

}

override int AdviseHierarchyEvents(IVsHierarchyEvents pEventSink, uint \*pdwCookie)

{

// use this as an callback of the project load being complete

if(mLastHierarchyEventSinkCookie == 0)

Package.scheduleUpdateLibrary();

return super.AdviseHierarchyEvents(pEventSink, pdwCookie);

}

// IVsGetCfgProvider

override int GetCfgProvider(IVsCfgProvider\* pCfgProvider)

{

//mixin(LogCallMix);

\*pCfgProvider = addref(mConfigProvider);

return S\_OK;

}

// ISpecifyPropertyPages

override int GetPages( /\* [out] \*/ CAUUID \*pPages)

{

// needs common properties to not open settings dialog modal

mixin(LogCallMix);

return PropertyPageFactory.GetCommonPages(pPages);

}

// IVsAggregatableProject

override int SetInnerProject(

/\* [in] \*/ IUnknown punkInner)

{

logCall("%S.SetInnerProject(punkInner=%s)", this, \_toLog(punkInner));

return returnError(E\_NOTIMPL);

}

override int InitializeForOuter(

/\* [in] \*/ in wchar\* pszFilename,

/\* [in] \*/ in wchar\* pszLocation,

/\* [in] \*/ in wchar\* pszName,

/\* [in] \*/ in VSCREATEPROJFLAGS grfCreateFlags,

/\* [in] \*/ in IID\* iidProject,

/\* [iid\_is][out] \*/ void \*\*ppvProject,

/\* [out] \*/ BOOL \*pfCanceled)

{

mixin(LogCallMix);

return returnError(E\_NOTIMPL);

}

override int OnAggregationComplete()

{

mixin(LogCallMix);

return returnError(E\_NOTIMPL);

}

override int GetAggregateProjectTypeGuids(

/\* [out] \*/ BSTR \*pbstrProjTypeGuids)

{

logCall("GetAggregateProjectTypeGuids(pbstrProjTypeGuids=%s)", \_toLog(pbstrProjTypeGuids));

wstring s = GUID2wstring(g\_projectFactoryCLSID);

\*pbstrProjTypeGuids = allocwBSTR(s);

return S\_OK;

}

override int SetAggregateProjectTypeGuids(

/\* [in] \*/ in wchar\* lpstrProjTypeGuids)

{

logCall("SetAggregateProjectTypeGuids(lpstrProjTypeGuids=%s)", \_toLog(lpstrProjTypeGuids));

return returnError(E\_NOTIMPL);

}

// IVsProjectFlavorCfgProvider

override int CreateProjectFlavorCfg(

/\* [in] \*/ IVsCfg pBaseProjectCfg,

/\* [out] \*/ IVsProjectFlavorCfg \*ppFlavorCfg)

{

mixin(LogCallMix);

return returnError(E\_NOTIMPL);

}

// IPersist

override int GetClassID(CLSID\* pClassID)

{

mixin(LogCallMix2);

\*cast(GUID\*)pClassID = g\_projectFactoryCLSID;

return S\_OK;

}

// IPersistFileFormat

override int IsDirty(

/\* [out] \*/ BOOL \*pfIsDirty)

{

logCall("IsDirty(pfIsDirty=%s)", \_toLog(pfIsDirty));

if(CProjectNode pProjectNode = GetProjectNode())

\*pfIsDirty = pProjectNode.IsProjectFileDirty();

else

return E\_FAIL;

return S\_OK;

}

override int InitNew(

/\* [in] \*/ in DWORD nFormatIndex)

{

logCall("InitNew(nFormatIndex=%s)", \_toLog(nFormatIndex));

// mixin(LogCallMix);

return returnError(E\_NOTIMPL);

}

override int Load(

/\* [in] \*/ in wchar\* pszFilename,

/\* [in] \*/ in DWORD grfMode,

/\* [in] \*/ in BOOL fReadOnly)

{

mixin(LogCallMix);

return returnError(E\_NOTIMPL);

}

override int Save(

/\* [in] \*/ in wchar\* pszFilename,

/\* [in] \*/ in BOOL fRemember,

/\* [in] \*/ in DWORD nFormatIndex)

{

mixin(LogCallMix);

auto doc = createDoc();

string filename = to\_string(pszFilename);

if(!saveXML(doc, filename))

return returnError(E\_FAIL);

return S\_OK;

}

override int SaveCompleted(

/\* [in] \*/ in wchar\* pszFilename)

{

logCall("SaveCompleted(pszFilename=%s)", \_toLog(pszFilename));

if(mFilename == to\_string(pszFilename)) // autosave?

if(CProjectNode pProjectNode = GetProjectNode())

pProjectNode.SetProjectFileDirty(false);

return S\_OK; //returnError(E\_NOTIMPL);

}

override int GetCurFile(

/\* [out] \*/ LPOLESTR \*ppszFilename,

/\* [out] \*/ DWORD \*pnFormatIndex)

{

mixin(LogCallMix);

\*ppszFilename = string2OLESTR(mFilename);

\*pnFormatIndex = 0;

return S\_OK;

}

override int GetFormatList(

/\* [out] \*/ LPOLESTR \*ppszFormatList)

{

logCall("GetFormatList(pbstrProjTypeGuids=%s)", \_toLog(ppszFormatList));

return returnError(E\_NOTIMPL);

}

// IVsProjectBuildSystem

override int SetHostObject(

/\* [in] \*/ in wchar\* pszTargetName,

/\* [in] \*/ in wchar\* pszTaskName,

/\* [in] \*/ IUnknown punkHostObject)

{

mixin(LogCallMix);

return returnError(E\_NOTIMPL);

}

override int StartBatchEdit()

{

mixin(LogCallMix);

return returnError(E\_NOTIMPL);

}

override int EndBatchEdit()

{

mixin(LogCallMix);

return returnError(E\_NOTIMPL);

}

override int CancelBatchEdit()

{

mixin(LogCallMix);

return returnError(E\_NOTIMPL);

}

override int BuildTarget(

/\* [in] \*/ in wchar\* pszTargetName,

/\* [retval][out] \*/ VARIANT\_BOOL \*pbSuccess)

{

mixin(LogCallMix);

return returnError(E\_NOTIMPL);

}

override int GetBuildSystemKind(

/\* [retval][out] \*/ BuildSystemKindFlags \*pBuildSystemKind)

{

//                mixin(LogCallMix);

\*pBuildSystemKind = 0;

return S\_OK;

}

// IVsBuildPropertyStorage

override int GetPropertyValue(

/\* [in] \*/ in wchar\* pszPropName,

/\* [in] \*/ in wchar\* pszConfigName,

/\* [in] \*/ in PersistStorageType storage,

/\* [retval][out] \*/ BSTR \*pbstrPropValue)

{

mixin(LogCallMix);

string prop = to\_string(pszPropName);

string value;

/+

if(prop == "RegisterOutputPackage")

value = "true";

+/

if(value.length == 0)

return DISP\_E\_MEMBERNOTFOUND;

\*pbstrPropValue = allocBSTR(value);

return S\_OK;

}

override int SetPropertyValue(

/\* [in] \*/ in wchar\* pszPropName,

/\* [in] \*/ in wchar\* pszConfigName,

/\* [in] \*/ in PersistStorageType storage,

/\* [in] \*/ in wchar\* pszPropValue)

{

mixin(LogCallMix);

return returnError(E\_NOTIMPL);

}

override int RemoveProperty(

/\* [in] \*/ in wchar\* pszPropName,

/\* [in] \*/ in wchar\* pszConfigName,

/\* [in] \*/ in PersistStorageType storage)

{

mixin(LogCallMix);

return returnError(E\_NOTIMPL);

}

override int GetItemAttribute(

/\* [in] \*/ in VSITEMID item,

/\* [in] \*/ in wchar\* pszAttributeName,

/\* [out] \*/ BSTR \*pbstrAttributeValue)

{

mixin(LogCallMix);

return returnError(E\_NOTIMPL);

}

override int SetItemAttribute(

/\* [in] \*/ in VSITEMID item,

/\* [in] \*/ in wchar\* pszAttributeName,

/\* [in] \*/ in wchar\* pszAttributeValue)

{

mixin(LogCallMix);

return returnError(E\_NOTIMPL);

}

// IVsComponentUser

override int AddComponent(

/\* [in] \*/ in VSADDCOMPOPERATION dwAddCompOperation,

/\* [in] \*/ in ULONG cComponents,

/\* [size\_is][in] \*/ in PVSCOMPONENTSELECTORDATA \*rgpcsdComponents,

/\* [in] \*/ in HWND hwndPickerDlg,

/\* [retval][out] \*/ VSADDCOMPRESULT \*pResult)

{

mixin(LogCallMix);

return returnError(E\_NOTIMPL);

}

// IVsDependencyProvider

override int EnumDependencies(

/\* [out] \*/ IVsEnumDependencies \*ppIVsEnumDependencies)

{

mixin(LogCallMix);

return returnError(E\_NOTIMPL);

}

override int OpenDependency(

/\* [in] \*/ in wchar\* szDependencyCanonicalName,

/\* [out] \*/ IVsDependency \*ppIVsDependency)

{

mixin(LogCallMix);

return returnError(E\_NOTIMPL);

}

// IVsProjectSpecialFiles

override int GetFile(

/\* [in] \*/ in PSFFILEID fileID,

/\* [in] \*/ in PSFFLAGS grfFlags,

/\* [out] \*/ VSITEMID \*pitemid,

/\* [out] \*/ BSTR \*pbstrFilename)

{

mixin(LogCallMix);

return returnError(E\_NOTIMPL);

}

// IVsParentProject

override int OpenChildren()

{

mixin(LogCallMix);

// config not yet known here

return returnError(E\_NOTIMPL);

}

override int CloseChildren()

{

mixin(LogCallMix);

return returnError(E\_NOTIMPL);

}

// CVsHierarchy

override HRESULT QueryStatusSelection(in GUID \*pguidCmdGroup,

in ULONG cCmds, OLECMD \*prgCmds, OLECMDTEXT \*pCmdText,

ref CHierNode[] rgSelection,

bool bIsHierCmd)// TRUE if cmd originated via CVSUiHierarchy::ExecCommand

{

assert(pguidCmdGroup);

assert(prgCmds);

assert(cCmds == 1);

HRESULT hr = S\_OK;

bool fHandled = false;

bool fSupported = false;

bool fEnabled = false;

bool fInvisible = false;

bool fLatched = false;

OLECMD \*Cmd = prgCmds;

if (\*pguidCmdGroup == CMDSETID\_StandardCommandSet97)

{

// NOTE: We only want to support Cut/Copy/Paste/Delete/Rename commands

// if focus is in the project window. This means that we should only

// support these commands if they are dispatched via IVsUIHierarchy

// interface and not if they are dispatch through IOleCommandTarget

// during the command routing to the active project/hierarchy.

if(!bIsHierCmd)

{

switch(Cmd.cmdID)

{

case cmdidCut:

case cmdidCopy:

case cmdidPaste:

case cmdidRename:

return OLECMDERR\_E\_NOTSUPPORTED;

default:

break;

}

}

switch(Cmd.cmdID)

{

// Forward the following commands to the project node whenever our project is

// the active project.

case cmdidAddNewItem:

case cmdidAddExistingItem:

case cmdidBuildSel:

case cmdidRebuildSel:

case cmdidCleanSel:

case cmdidCancelBuild:

case cmdidProjectSettings:

case cmdidBuildSln:

case cmdidUnloadProject:

case cmdidSetStartupProject:

return GetProjectNode().QueryStatus(pguidCmdGroup, cCmds, prgCmds, pCmdText);

default:

break;

}

}

else if(\*pguidCmdGroup == CMDSETID\_StandardCommandSet2K)

{

switch(Cmd.cmdID)

{

case cmdidBuildOnlyProject:

case cmdidRebuildOnlyProject:

case cmdidCleanOnlyProject:

return GetProjectNode().QueryStatus(pguidCmdGroup, cCmds, prgCmds, pCmdText);

case ECMD\_SHOWALLFILES:

debug

{

Cmd.cmdf = OLECMDF\_SUPPORTED | OLECMDF\_ENABLED;

return hr;

}

default:

break;

}

}

else if(\*pguidCmdGroup == g\_commandSetCLSID)

{

switch(Cmd.cmdID)

{

case CmdNewPackage:

case CmdNewFilter:

return GetProjectNode().QueryStatus(pguidCmdGroup, cCmds, prgCmds, pCmdText);

default:

break;

}

}

// Node commands

if (!fHandled)

{

fHandled = true;

OLECMD cmdTemp;

cmdTemp.cmdID = Cmd.cmdID;

fSupported = false;

fEnabled = true;

fInvisible = false;

fLatched = true;

foreach (pNode; rgSelection)

{

cmdTemp.cmdf = 0;

hr = pNode.QueryStatus(pguidCmdGroup, 1, &cmdTemp, pCmdText);

if (SUCCEEDED(hr))

{

//

// cmd is supported iff any node supports cmd

// cmd is enabled iff all nodes enable cmd

// cmd is invisible iff any node sets invisibility

// cmd is latched only if all are latched.

fSupported = fSupported || (cmdTemp.cmdf & OLECMDF\_SUPPORTED);

fEnabled = fEnabled && (cmdTemp.cmdf & OLECMDF\_ENABLED);

fInvisible = fInvisible || (cmdTemp.cmdf & OLECMDF\_INVISIBLE);

fLatched = fLatched && (cmdTemp.cmdf & OLECMDF\_LATCHED);

//NOTE: Currently no commands use NINCHED

assert(!(cmdTemp.cmdf & OLECMDF\_NINCHED));

}

// optimization

if (!fSupported || fInvisible)

break;

}

}

if (SUCCEEDED(hr) && fSupported)

{

Cmd.cmdf = OLECMDF\_SUPPORTED;

if (fEnabled)

Cmd.cmdf |= OLECMDF\_ENABLED;

if (fInvisible)

Cmd.cmdf |= OLECMDF\_INVISIBLE;

if (fLatched)

Cmd.cmdf |= OLECMDF\_LATCHED;

}

return hr;

}

// IVsGlobalsCallback

override int WriteVariablesToData(

/\* [in] \*/ in wchar\* pVariableName,

/\* [in] \*/ in VARIANT \*varData)

{

mixin(LogCallMix);

return returnError(E\_NOTIMPL);

}

override int ReadData(/\* [in] \*/ dte.Globals pGlobals)

{

logCall("%s.ReadData(pGlobals=%s)", this, \_toLog(pGlobals));

return returnError(E\_NOTIMPL);

}

override int ClearVariables()

{

mixin(LogCallMix);

return returnError(E\_NOTIMPL);

}

override int VariableChanged()

{

mixin(LogCallMix);

return returnError(E\_NOTIMPL);

}

override int CanModifySource()

{

mixin(LogCallMix);

return returnError(E\_NOTIMPL);

}

override int GetParent(IDispatch \*ppOut)

{

logCall("%s.GetParent()", this);

return returnError(E\_NOTIMPL);

}

// IPerPropertyBrowsing

override int GetDisplayString(

/\* [in] \*/ in DISPID dispID,

/\* [out] \*/ BSTR \*pBstr)

{

logCall("%s.GetDisplayString(dispID=%s, pBstr=%s)", this, \_toLog(dispID), \_toLog(pBstr));

return returnError(E\_NOTIMPL);

}

override int MapPropertyToPage(

/\* [in] \*/ in DISPID dispID,

/\* [out] \*/ CLSID \*pClsid)

{

mixin(LogCallMix);

\*cast(GUID\*)pClsid = g\_GeneralPropertyPage;

return S\_OK;

//return returnError(E\_NOTIMPL);

}

override int GetPredefinedStrings(

/\* [in] \*/ in DISPID dispID,

/\* [out] \*/ CALPOLESTR \*pCaStringsOut,

/\* [out] \*/ CADWORD \*pCaCookiesOut)

{

mixin(LogCallMix);

return returnError(E\_NOTIMPL);

}

override int GetPredefinedValue(

/\* [in] \*/ in DISPID dispID,

/\* [in] \*/ in DWORD dwCookie,

/\* [out] \*/ VARIANT \*pVarOut)

{

mixin(LogCallMix);

return returnError(E\_NOTIMPL);

}

// IVsNonLocalProject

override HRESULT EnsureLocalCopy(in VSITEMID itemid)

{

logCall("%s.EnsureLocalCopy(this=%s, itemid=%x)", this, cast(void\*)this, itemid);

return S\_OK;

}

/+

// IRpcOptions

override HRESULT Set(/+[in]+/ IUnknown pPrx, in DWORD dwProperty, in ULONG\_PTR dwValue)

{

mixin(LogCallMix);

return E\_NOTIMPL;

}

override HRESULT Query(/+[in]+/ IUnknown pPrx, in DWORD dwProperty, /+[out]+/ ULONG\_PTR \* pdwValue)

{

mixin(LogCallMix);

if(dwProperty == COMBND\_RPCTIMEOUT)

\*pdwValue = RPC\_C\_BINDING\_MAX\_TIMEOUT;

else if(dwProperty == COMBND\_SERVER\_LOCALITY)

\*pdwValue = SERVER\_LOCALITY\_PROCESS\_LOCAL;

else

return E\_NOTIMPL;

return S\_OK;

}

+/

// IVsSccProject2

override HRESULT SccGlyphChanged(in int cAffectedNodes,

/+[size\_is(cAffectedNodes)]+/in VSITEMID \*rgitemidAffectedNodes,

/+[size\_is(cAffectedNodes)]+/in VsStateIcon \*rgsiNewGlyphs,

/+[size\_is(cAffectedNodes)]+/in DWORD \*rgdwNewSccStatus)

{

mixin(LogCallMix);

if(cAffectedNodes == 0)

{

searchNode(GetRootNode(), delegate (CHierNode n)

{

foreach (advise; mHierarchyEventSinks)

advise.OnPropertyChanged(GetVsItemID(n), VSHPROPID\_StateIconIndex, 0);

return false;

});

}

else

{

for(int i = 0; i < cAffectedNodes; i++)

foreach (advise; mHierarchyEventSinks)

advise.OnPropertyChanged(rgitemidAffectedNodes[i], VSHPROPID\_StateIconIndex, 0);

}

return S\_OK;

}

override HRESULT SetSccLocation(in LPCOLESTR pszSccProjectName, // opaque to project

in LPCOLESTR pszSccAuxPath, // opaque to project

in LPCOLESTR pszSccLocalPath, // opaque to project

in LPCOLESTR pszSccProvider) // opaque to project

{

mixin(LogCallMix);

return E\_NOTIMPL;

}

override HRESULT GetSccFiles(in VSITEMID itemid, // Node in project hierarchy

/+[out]+/ CALPOLESTR \*pCaStringsOut, // Files associated with node

/+[out]+/ CADWORD \*pCaFlagsOut) // Flags per file

{

mixin(LogCallMix);

CHierNode node = VSITEMID2Node(itemid);

if(node)

{

pCaStringsOut.pElems = cast(wchar\*\*) CoTaskMemAlloc(pCaStringsOut.pElems[0].sizeof);

pCaStringsOut.cElems = 1;

pCaStringsOut.pElems[0] = string2OLESTR(node.GetFullPath());

pCaFlagsOut.pElems = cast(uint\*) CoTaskMemAlloc(pCaFlagsOut.pElems[0].sizeof);

pCaFlagsOut.cElems = 1;

pCaFlagsOut.pElems[0] = SFF\_NoFlags;

logCall(" %s.GetSccFiles returns %s", this, \_toLog(pCaStringsOut.pElems[0]));

return S\_OK;

}

return S\_FALSE;

}

override HRESULT GetSccSpecialFiles(in VSITEMID itemid, // node in project hierarchy

in LPCOLESTR pszSccFile, // one of the files associated with the node

/+[out]+/ CALPOLESTR \*pCaStringsOut, // special files associated with above file

/+[out]+/ CADWORD \*pCaFlagsOut) // flags per special file

{

mixin(LogCallMix);

return E\_NOTIMPL;

}

///////////////////////////////////////////////////////////////////////

// IVsHierarchyDropDataSource

override int GetDropInfo(

/\* [out] \*/ DWORD \*pdwOKEffects,

/\* [out] \*/ IDataObject \*ppDataObject,

/\* [out] \*/ IDropSource \*ppDropSource)

{

mixin(LogCallMix);

\*pdwOKEffects = DROPEFFECT\_NONE;

\*ppDataObject = null;

\*ppDropSource = null;

HRESULT hr = PackageSelectionDataObject(ppDataObject, FALSE);

if(FAILED(hr))

return returnError(hr);

\*pdwOKEffects = DROPEFFECT\_MOVE | DROPEFFECT\_COPY;

mDDT = DropDataType.DDT\_VSREF;

mfDragSource = TRUE;

return S\_OK;

}

override int OnDropNotify(

/\* [in] \*/ in BOOL fDropped,

/\* [in] \*/ in DWORD dwEffects)

{

mixin(LogCallMix);

mfDragSource = FALSE;

mDDT = DropDataType.DDT\_NONE;

return CleanupSelectionDataObject(fDropped, FALSE, dwEffects == DROPEFFECT\_MOVE);

}

// IVsHierarchyDropDataSource2

override int OnBeforeDropNotify(

/\* [in] \*/ IDataObject pDataObject,

/\* [in] \*/ in DWORD dwEffect,

/\* [retval][out] \*/ BOOL \*pfCancelDrop)

{

mixin(LogCallMix);

if (pfCancelDrop)

\*pfCancelDrop = FALSE;

HRESULT hr = S\_OK;

// check for dirty documents

BOOL fDirty = FALSE;

for (ULONG i = 0; i < mItemSelDragged.length; i++)

{

CFileNode pFileNode = cast(CFileNode) VSITEMID2Node(mItemSelDragged[i].itemid);

if (!pFileNode)

continue;

bool fDirtyDoc = FALSE;

bool fOpenByUs = FALSE;

hr = pFileNode.GetDocInfo(

/\* [out, opt] BOOL\* pfOpen \*/ null, // true if the doc is opened

/\* [out, opt] BOOL\* pfDirty \*/ &fDirtyDoc, // true if the doc is dirty

/\* [out, opt] BOOL\* pfOpenByUs \*/ &fOpenByUs, // true if opened by our project

/\* [out, opt] VSDOCCOOKIE\* pVsDocCookie\*/ null);// VSDOCCOOKIE if open

if (FAILED(hr))

continue;

if (fDirtyDoc && fOpenByUs)

{

fDirty = TRUE;

break;

}

}

// if there are no dirty docs we are ok to proceed

if (!fDirty)

return S\_OK;

// prompt to save if there are dirty docs

string caption = "Visual Studio D'n'D";

string prompt = "Save modified documents?";

int msgRet = UtilMessageBox(prompt, MB\_YESNOCANCEL | MB\_ICONEXCLAMATION, caption);

switch (msgRet)

{

case IDYES:

break;

case IDNO:

return S\_OK;

case IDCANCEL:

if (pfCancelDrop)

\*pfCancelDrop = TRUE;

return S\_OK;

default:

assert(\_false);

return S\_OK;

}

for (ULONG i = 0; i < mItemSelDragged.length; i++)

{

if(CFileNode pFileNode = cast(CFileNode) VSITEMID2Node(mItemSelDragged[i].itemid))

hr = pFileNode.SaveDoc(SLNSAVEOPT\_SaveIfDirty);

}

return returnError(hr);

}

// IVsHierarchyDropDataTarget

override int DragEnter(

/\* [in] \*/ IDataObject pDataObject,

/\* [in] \*/ in DWORD grfKeyState,

/\* [in] \*/ in VSITEMID itemid,

/\* [out][in] \*/ DWORD \*pdwEffect)

{

mixin(LogCallMix);

\*pdwEffect = DROPEFFECT\_NONE;

if (mfDragSource)

return S\_OK;

if(HRESULT hr = QueryDropDataType(pDataObject))

return hr;

return QueryDropEffect(mDDT, grfKeyState, pdwEffect);

}

override int DragOver(

/\* [in] \*/ in DWORD grfKeyState,

/\* [in] \*/ in VSITEMID itemid,

/\* [out][in] \*/ DWORD \*pdwEffect)

{

mixin(LogCallMix);

return QueryDropEffect(mDDT, grfKeyState, pdwEffect);

}

override int DragLeave()

{

mixin(LogCallMix);

if (!mfDragSource)

mDDT = DropDataType.DDT\_NONE;

return S\_OK;

}

override int Drop(

/\* [in] \*/ IDataObject pDataObject,

/\* [in] \*/ in DWORD grfKeyState,

/\* [in] \*/ in VSITEMID itemid,

/\* [out][in] \*/ DWORD \*pdwEffect)

{

mixin(LogCallMix);

if (!pDataObject)

return E\_INVALIDARG;

if (!pdwEffect)

return E\_POINTER;

\*pdwEffect = DROPEFFECT\_NONE;

HRESULT hr = S\_OK;

//                if (mfDragSource)

//                        return S\_OK;

CHierNode dropNode = VSITEMID2Node(itemid);

if(!dropNode)

dropNode = GetProjectNode();

CHierContainer dropContainer = cast(CHierContainer) dropNode;

if(!dropContainer)

dropContainer = dropNode.GetParent();

DropDataType ddt;

hr = ProcessSelectionDataObject(dropContainer,

/\* [in] IDataObject\* pDataObject\*/ pDataObject,

/\* [in] DWORD grfKeyState\*/ grfKeyState,

/\* [out] DropDataType\* \*/ &ddt);

// We need to report our own errors.

if(FAILED(hr) && hr != E\_UNEXPECTED && hr != OLE\_E\_PROMPTSAVECANCELLED)

{

UtilReportErrorInfo(hr);

}

// If it is a drop from windows and we get any kind of error we return S\_FALSE and dropeffect none. This

// prevents bogus messages from the shell from being displayed

if(FAILED(hr) && ddt == DropDataType.DDT\_SHELL)

{

hr = S\_FALSE;

}

if (hr == S\_OK)

QueryDropEffect(ddt, grfKeyState, pdwEffect);

return hr;

}

enum DropDataType //Drop types

{

DDT\_NONE,

DDT\_SHELL,

DDT\_VSSTG,

DDT\_VSREF

};

enum ushort CF\_HDROP = 15; // winuser.h

int QueryDropDataType(IDataObject pDataObject)

{

mDDT = DropDataType.DDT\_NONE;

// known formats include File Drops (as from WindowsExplorer),

// VSProject Reference Items and VSProject Storage Items.

FORMATETC fmtetc, fmtetcRef, fmtetcStg;

fmtetc.cfFormat = CF\_HDROP;

fmtetc.ptd = null;

fmtetc.dwAspect = DVASPECT\_CONTENT;

fmtetc.lindex = -1;

fmtetc.tymed = TYMED\_HGLOBAL;

fmtetcRef.cfFormat = cast(CLIPFORMAT) RegisterClipboardFormatW("CF\_VSREFPROJECTITEMS"w.ptr);

fmtetcRef.ptd = null;

fmtetcRef.dwAspect = DVASPECT\_CONTENT;

fmtetcRef.lindex = -1;

fmtetcRef.tymed = TYMED\_HGLOBAL;

fmtetcStg.cfFormat = cast(CLIPFORMAT) RegisterClipboardFormatW("CF\_VSSTGPROJECTITEMS"w.ptr);

fmtetcStg.ptd = null;

fmtetcStg.dwAspect = DVASPECT\_CONTENT;

fmtetcStg.lindex = -1;

fmtetcStg.tymed = TYMED\_HGLOBAL;

if (pDataObject.QueryGetData(&fmtetc) == S\_OK)

{

mDDT = DropDataType.DDT\_SHELL;

return S\_OK;

}

if (pDataObject.QueryGetData(&fmtetcRef) == S\_OK)

{

// Data is from a Ref-based project.

mDDT = DropDataType.DDT\_VSREF;

return S\_OK;

}

if (pDataObject.QueryGetData(&fmtetcStg) == S\_OK)

{

// Data is from a Storage-based project.

mDDT = DropDataType.DDT\_VSSTG;

return S\_OK;

}

return S\_FALSE;

}

int QueryDropEffect(

/\* [in] \*/ DropDataType ddt,

/\* [in] \*/ DWORD grfKeyState,

/\* [out] \*/ DWORD \* pdwEffects)

{

\*pdwEffects = DROPEFFECT\_NONE;

HRESULT hr = S\_OK;

// We are reference-based project so we should perform as follow:

// for shell and physical items:

// NO MODIFIER - LINK

// SHIFT DRAG - NO DROP

// CTRL DRAG - NO DROP

// CTRL-SHIFT DRAG - LINK

// for reference/link items

// NO MODIFIER - MOVE

// SHIFT DRAG - MOVE

// CTRL DRAG - COPY

// CTRL-SHIFT DRAG - LINK

if(ddt != DropDataType.DDT\_SHELL && ddt != DropDataType.DDT\_VSREF && ddt != DropDataType.DDT\_VSSTG)

return S\_FALSE;

switch (ddt)

{

case DropDataType.DDT\_SHELL:

case DropDataType.DDT\_VSSTG:

// CTRL-SHIFT

if((grfKeyState & MK\_CONTROL) && (grfKeyState & MK\_SHIFT))

{

\*pdwEffects = DROPEFFECT\_LINK;

return S\_OK;

}

// CTRL

if(grfKeyState & MK\_CONTROL)

return S\_FALSE;

// SHIFT

if(grfKeyState & MK\_SHIFT)

return S\_FALSE;

// no modifier

\*pdwEffects = DROPEFFECT\_LINK;

return S\_OK;

case DropDataType.DDT\_VSREF:

// CTRL-SHIFT

if((grfKeyState & MK\_CONTROL) && (grfKeyState & MK\_SHIFT))

{

\*pdwEffects = DROPEFFECT\_LINK;

return S\_OK;

}

// CTRL

if(grfKeyState & MK\_CONTROL)

{

\*pdwEffects = DROPEFFECT\_COPY;

return S\_OK;

}

// SHIFT

if(grfKeyState & MK\_SHIFT)

{

\*pdwEffects = DROPEFFECT\_MOVE;

return S\_OK;

}

// no modifier

\*pdwEffects = DROPEFFECT\_MOVE;

return S\_OK;

default:

return S\_FALSE;

}

}

bool isChildItem(CHierContainer dropTarget, IVsHierarchy srpIVsHierarchy, VSITEMID itemidLoc)

{

if(srpIVsHierarchy !is this)

return false;

CHierNode dropSource = VSITEMID2Node(itemidLoc);

for(CHierNode c = dropTarget; c; c = c.GetParent())

if(dropSource == c)

return true;

return false;

}

HRESULT copyVirtualFolder(CHierContainer dropContainer, IVsHierarchy srpIVsHierarchy, VSITEMID itemidLoc)

{

if(isChildItem(dropContainer, srpIVsHierarchy, itemidLoc))

{

UtilMessageBox("Cannot drop folder into itself or one of its sub folders", MB\_OK, "Drop folder");

return S\_FALSE;

}

IVsProject srpIVsProject = qi\_cast!IVsProject(srpIVsHierarchy);

if(!srpIVsProject)

return E\_UNEXPECTED;

scope(exit) release(srpIVsProject);

BSTR cbstrMoniker;

if(HRESULT hr = srpIVsProject.GetMkDocument(itemidLoc, &cbstrMoniker))

return hr;

string name = detachBSTR(cbstrMoniker);

CFolderNode pFolder = newCom!CFolderNode;

string strThisFolder = baseName(name);

pFolder.SetName(strThisFolder);

VARIANT var;

if(srpIVsHierarchy.GetProperty(itemidLoc, VSHPROPID\_FirstChild, &var) == S\_OK &&

(var.vt == VT\_INT\_PTR || var.vt == VT\_I4 || var.vt == VT\_INT))

{

VSITEMID chid = var.lVal;

while(chid != VSITEMID\_NIL)

{

if(HRESULT hr = processVSItem(pFolder, srpIVsHierarchy, chid))

return hr;

if(srpIVsHierarchy.GetProperty(chid, VSHPROPID\_NextSibling, &var) != S\_OK ||

(var.vt != VT\_INT\_PTR && var.vt != VT\_I4 && var.vt != VT\_INT))

break;

chid = var.lVal;

}

}

dropContainer.Add(pFolder);

return S\_OK;

}

HRESULT processVSItem(CHierContainer dropContainer, IVsHierarchy srpIVsHierarchy, VSITEMID itemidLoc)

{

// If this is a virtual item, we skip it

GUID typeGuid;

bool isFolder = false;

HRESULT hr = srpIVsHierarchy.GetGuidProperty(itemidLoc, VSHPROPID\_TypeGuid, &typeGuid);

if(SUCCEEDED(hr) && typeGuid == GUID\_ItemType\_VirtualFolder)

return copyVirtualFolder(dropContainer, srpIVsHierarchy, itemidLoc);

if(SUCCEEDED(hr) && typeGuid != GUID\_ItemType\_PhysicalFile)

return S\_FALSE;

if(hr == E\_ABORT || hr == OLE\_E\_PROMPTSAVECANCELLED)

return OLE\_E\_PROMPTSAVECANCELLED;

IVsProject srpIVsProject;

scope(exit) release(srpIVsProject);

hr = srpIVsHierarchy.QueryInterface(&IVsProject.iid, cast(void \*\*)&srpIVsProject);

if(FAILED(hr) || !srpIVsProject)

return hr;

BSTR cbstrMoniker;

hr = srpIVsProject.GetMkDocument(itemidLoc, &cbstrMoniker);

if (FAILED(hr))

return hr;

string filename = detachBSTR(cbstrMoniker);

wchar\* wfilename = \_toUTF16z(filename);

VSADDRESULT vsaddresult = ADDRESULT\_Failure;

hr = GetProjectNode().GetCVsHierarchy().AddItemSpecific(dropContainer,

/\* [in] VSADDITEMOPERATION dwAddItemOperation \*/ VSADDITEMOP\_OPENFILE,

/\* [in] LPCOLESTR pszItemName \*/ null,

/\* [in] DWORD cFilesToOpen \*/ 1,

/\* [in] LPCOLESTR rgpszFilesToOpen[] \*/ &wfilename,

/\* [in] HWND hwndDlg \*/ null,

/\* [in] VSSPECIFICEDITORFLAGS grfEditorFlags \*/ cast(VSSPECIFICEDITORFLAGS) 0,

/\* [in] REFGUID rguidEditorType \*/ &GUID\_NULL,

/\* [in] LPCOLESTR pszPhysicalView \*/ null,

/\* [in] REFGUID rguidLogicalView\*/ &GUID\_NULL,

/\* [in] bool moveIfInProject \*/ mfDragSource,

/\* [out] VSADDRESULT \*pResult \*/ &vsaddresult);

if (hr == E\_ABORT || hr == OLE\_E\_PROMPTSAVECANCELLED || vsaddresult == ADDRESULT\_Cancel)

return OLE\_E\_PROMPTSAVECANCELLED;

return hr;

}

HRESULT ProcessSelectionDataObject(

/\* [in] \*/ CHierContainer dropContainer,

/\* [in] \*/ IDataObject pDataObject,

/\* [in] \*/ DWORD grfKeyState,

/\* [out] \*/ DropDataType\* pddt)

{

HRESULT hr = S\_OK;

if (pddt)

\*pddt = DropDataType.DDT\_NONE;

CProjectNode pProjectNode = GetProjectNode();

FORMATETC fmtetc;

STGMEDIUM stgmedium;

HANDLE hDropInfo = null;

int numFiles = 0;

wchar[MAX\_PATH+1] szMoniker;

DropDataType ddt = DropDataType.DDT\_NONE;

BOOL fItemProcessed = FALSE;

// try HDROP

fmtetc.cfFormat = CF\_HDROP;

fmtetc.ptd = null;

fmtetc.dwAspect = DVASPECT\_CONTENT;

fmtetc.lindex = -1;

fmtetc.tymed = TYMED\_HGLOBAL;

if(pDataObject.QueryGetData(&fmtetc) != S\_OK ||

FAILED(pDataObject.GetData(&fmtetc, &stgmedium)) ||

stgmedium.tymed != TYMED\_HGLOBAL || !stgmedium.hGlobal)

goto AttemptVSRefFormat;

hDropInfo = stgmedium.hGlobal;

// try shell format here

ddt = DropDataType.DDT\_SHELL;

numFiles = .DragQueryFileW(hDropInfo, 0xFFFFFFFF, null, 0);

for (int iFile = 0; iFile < numFiles; iFile++)

{

UINT uiRet = .DragQueryFileW(hDropInfo, iFile, szMoniker.ptr, \_MAX\_PATH);

if (!uiRet || uiRet >= \_MAX\_PATH)

{

hr = E\_OUTOFMEMORY; // HRESULT\_FROM\_WIN32(ERROR\_INSUFFICIENT\_BUFFER);

continue;

}

szMoniker[\_MAX\_PATH] = 0;

string filename = to\_string(szMoniker.ptr);

// Is full path returned

if (exists(filename))

{

VSADDRESULT vsaddresult = ADDRESULT\_Failure;

wchar\* wfilename = \_toUTF16z(filename);

HRESULT hrTemp = pProjectNode.GetCVsHierarchy().AddItemSpecific(dropContainer,

/\* [in] VSADDITEMOPERATION dwAddItemOperation \*/ VSADDITEMOP\_OPENFILE,

/\* [in] LPCOLESTR pszItemName \*/ null,

/\* [in] DWORD cFilesToOpen \*/ 1,

/\* [in] LPCOLESTR rgpszFilesToOpen[] \*/ &wfilename,

/\* [in] HWND hwndDlg \*/ null,

/\* [in] VSSPECIFICEDITORFLAGS grfEditorFlags \*/ cast(VSSPECIFICEDITORFLAGS) 0,

/\* [in] REFGUID rguidEditorType \*/ &GUID\_NULL,

/\* [in] LPCOLESTR pszPhysicalView \*/ null,

/\* [in] REFGUID rguidLogicalView\*/ &GUID\_NULL,

/\* [in] bool moveIfInProject \*/ mfDragSource,

/\* [out] VSADDRESULT \*pResult \*/ &vsaddresult);

if ( (hrTemp == E\_ABORT) || (hrTemp == OLE\_E\_PROMPTSAVECANCELLED) || (vsaddresult == ADDRESULT\_Cancel) )

{

hr = OLE\_E\_PROMPTSAVECANCELLED;

goto Error;

}

if (FAILED(hrTemp))

{

hr = hrTemp;

continue;

}

fItemProcessed = TRUE;

}

}

goto Error;

AttemptVSRefFormat:

fmtetc.cfFormat = cast(CLIPFORMAT) RegisterClipboardFormatW("CF\_VSREFPROJECTITEMS"w.ptr);

fmtetc.ptd = null;

fmtetc.dwAspect = DVASPECT\_CONTENT;

fmtetc.lindex = -1;

fmtetc.tymed = TYMED\_HGLOBAL;

if(pDataObject.QueryGetData(&fmtetc) != S\_OK ||

pDataObject.GetData(&fmtetc, &stgmedium) != S\_OK ||

stgmedium.tymed != TYMED\_HGLOBAL || !stgmedium.hGlobal)

goto AttemptVSStgFormat;

hDropInfo = stgmedium.hGlobal;

ddt = DropDataType.DDT\_VSREF;

goto AddFiles;

AttemptVSStgFormat:

fmtetc.cfFormat = cast(CLIPFORMAT) RegisterClipboardFormatW("CF\_VSSTGPROJECTITEMS"w.ptr);

fmtetc.ptd = null;

fmtetc.dwAspect = DVASPECT\_CONTENT;

fmtetc.lindex = -1;

fmtetc.tymed = TYMED\_HGLOBAL;

if(pDataObject.QueryGetData(&fmtetc) != S\_OK ||

pDataObject.GetData(&fmtetc, &stgmedium) != S\_OK ||

stgmedium.tymed != TYMED\_HGLOBAL || !stgmedium.hGlobal)

goto Error;

hDropInfo = stgmedium.hGlobal;

ddt = DropDataType.DDT\_VSSTG;

AddFiles:

if(IVsSolution srpIVsSolution = queryService!(IVsSolution))

{

scope(exit) release(srpIVsSolution);

// Note that we do NOT use ::DragQueryFile as this function will

// NOT work with unicode strings on win9x - even

// with the unicode wrappers - and the projitem ref format is in unicode

string[] rgSrcFiles;

numFiles = UtilGetFilesFromPROJITEMDrop(hDropInfo, rgSrcFiles);

for(int iFile = 0; iFile < numFiles; iFile++)

{

HRESULT hrTemp;

VSITEMID itemidLoc;

IVsHierarchy srpIVsHierarchy;

scope(exit) release(srpIVsHierarchy);

hrTemp = srpIVsSolution.GetItemOfProjref(\_toUTF16z(rgSrcFiles[iFile]), &srpIVsHierarchy, &itemidLoc, null, null);

if(hrTemp == E\_ABORT || hrTemp == OLE\_E\_PROMPTSAVECANCELLED)

{

hr = OLE\_E\_PROMPTSAVECANCELLED;

goto Error;

}

if (FAILED(hrTemp))

{

hr = hrTemp;

continue;

}

if (srpIVsHierarchy is null)

{

hr = E\_UNEXPECTED;

continue;

}

hr = processVSItem(dropContainer, srpIVsHierarchy, itemidLoc);

if(FAILED(hr))

goto Error;

if(hr == S\_OK)

fItemProcessed = TRUE;

}

}

Error:

if (hDropInfo)

.GlobalFree(hDropInfo);

if(FAILED(hr))

return hr;

if (!fItemProcessed || ddt == DropDataType.DDT\_NONE)

return S\_FALSE;

if (pddt)

\*pddt = ddt;

return S\_OK;

}

HRESULT PackageSelectionDataObject(

/\* [out] \*/ IDataObject \* ppDataObject,

/\* [in] \*/ BOOL fCutHighlightItems)

{

HRESULT hr = S\_OK;

// delete any existing selection data object and restore state

hr = CleanupSelectionDataObject(FALSE, FALSE, FALSE);

if(FAILED(hr)) return hr;

//                CComPtr<IVsUIHierarchyWindow> srpIVsUIHierarchyWindow;

//                hr = \_VxModule.GetIVsUIHierarchyWindow(GUID\_SolutionExplorer, &srpIVsUIHierarchyWindow);

//                IfFailRet(hr);

//                ExpectedExprRet(srpIVsUIHierarchyWindow != null);

IVsSolution srpIVsSolution = queryService!(IVsSolution);

if(!srpIVsSolution) return E\_NOINTERFACE;

scope(exit) release(srpIVsSolution);

IVsMonitorSelection srpIVsMonitorSelection = queryService!(IVsMonitorSelection);

if(!srpIVsMonitorSelection) return E\_NOINTERFACE;

scope(exit) release(srpIVsMonitorSelection);

VSITEMID vsitemid;

IVsHierarchy srpIVsHierarchy\_selection;

IVsMultiItemSelect srpIVsMultiItemSelect;

hr = srpIVsMonitorSelection.GetCurrentSelection(

/\* [out] IVsHierarchy\*\* \*/ &srpIVsHierarchy\_selection,

/\* [out] VSITEMID\* \*/ &vsitemid,

/\* [out] IVsMultiItemSelect\*\* \*/ &srpIVsMultiItemSelect,

/\* [out] ISelectionContainer\*\* \*/ null);

if(FAILED(hr)) return hr;

scope(exit) release(srpIVsHierarchy\_selection);

scope(exit) release(srpIVsMultiItemSelect);

LONG lLenGlobal = 0; // length of the file names including null chars

IVsHierarchy srpIVsHierarchy\_this = this; // GetIVsHierarchy();

if(srpIVsHierarchy\_selection !is srpIVsHierarchy\_this ||

vsitemid == VSITEMID\_ROOT || vsitemid == VSITEMID\_NIL)

return E\_ABORT;

if(vsitemid == VSITEMID\_SELECTION && srpIVsMultiItemSelect)

{

BOOL fSingleHierarchy = FALSE;

ULONG itemsDragged;

hr = srpIVsMultiItemSelect.GetSelectionInfo(&itemsDragged, &fSingleHierarchy);

if(FAILED(hr)) return hr;

if (!fSingleHierarchy) return E\_ABORT;

if (itemsDragged > uint.max / VSITEMSELECTION.sizeof)

return E\_OUTOFMEMORY;

mItemSelDragged.length = itemsDragged;

hr = srpIVsMultiItemSelect.GetSelectedItems(GSI\_fOmitHierPtrs, itemsDragged, mItemSelDragged.ptr);

if(FAILED(hr)) return hr;

}

else if (vsitemid != VSITEMID\_ROOT)

{

mItemSelDragged.length = 1;

mItemSelDragged[0].pHier = null;

mItemSelDragged[0].itemid = vsitemid;

}

for (ULONG i = 0; i < mItemSelDragged.length; i++)

{

if (mItemSelDragged[i].itemid == VSITEMID\_ROOT)

return E\_ABORT;

BSTR cbstrProjref;

hr = srpIVsSolution.GetProjrefOfItem(srpIVsHierarchy\_this, mItemSelDragged[i].itemid, &cbstrProjref);

if(FAILED(hr)) return hr;

wstring pref = wdetachBSTR(cbstrProjref);

if(pref.length==0)

return E\_FAIL;

lLenGlobal += pref.length + 1; // plus one to count the trailing null character

}

if(lLenGlobal == 0)

return E\_ABORT;

lLenGlobal += 1; // anothr trailing null character to terminate list

DWORD cbAlloc = DROPFILES.sizeof + lLenGlobal \* WCHAR.sizeof;// bytes to allocate

HGLOBAL hGlobal = GlobalAlloc(GHND | GMEM\_SHARE, cbAlloc);

if(!hGlobal) return E\_ABORT;

DROPFILES\* pDropFiles = cast(DROPFILES\*) GlobalLock(hGlobal);

// set the offset where the starting point of the file start

pDropFiles.pFiles = DROPFILES.sizeof;

// structure contain wide characters

pDropFiles.fWide = TRUE;

LPWSTR pFiles = cast(LPWSTR)(pDropFiles + 1);

LONG nCurPos = 0;

for (ULONG i = 0; i < mItemSelDragged.length; i++)

{

BSTR cbstrProjref;

hr = srpIVsSolution.GetProjrefOfItem(srpIVsHierarchy\_this, mItemSelDragged[i].itemid, &cbstrProjref);

if (FAILED(hr))

continue;

UINT cchProjRef = wcslen(cbstrProjref) + 1;

memcpy(pFiles + nCurPos, cbstrProjref, cchProjRef \* WCHAR.sizeof);

nCurPos += cchProjRef;

freeBSTR(cbstrProjref);

}

hr = S\_OK;

// final null terminator as per CF\_VSSTGPROJECTITEMS format spec

pFiles[nCurPos] = 0;

int res = GlobalUnlock(hGlobal);

OleDataSource pDataObject = newCom!OleDataSource; // has ref count of 0

FORMATETC fmtetc;

fmtetc.ptd = null;

fmtetc.dwAspect = DVASPECT\_CONTENT;

fmtetc.lindex = -1;

fmtetc.tymed = TYMED\_HGLOBAL;

fmtetc.cfFormat = cast(ushort) CF\_VSREFPROJECTITEMS();

STGMEDIUM stgmedium;

stgmedium.tymed = TYMED\_HGLOBAL;

stgmedium.hGlobal = hGlobal;

stgmedium.pUnkForRelease = null;

pDataObject.CacheData(fmtetc.cfFormat, &stgmedium, &fmtetc);

\*ppDataObject = addref(pDataObject);

Error:

/+

if (SUCCEEDED(hr))

{

if (fCutHighlightItems)

{

for (ULONG i = 0; i < mItemSelDragged.length; i++)

srpIVsUIHierarchyWindow.ExpandItem(GetIVsUIHierarchy(), mItemSelDragged[i].itemid, i == 0 ? EXPF\_CutHighlightItem : EXPF\_AddCutHighlightItem);

}

}

+/

if (FAILED(hr))

{

mItemSelDragged.length = 0;

}

return hr;

}

HRESULT CleanupSelectionDataObject(

/\* [in] \*/ BOOL fDropped,

/\* [in] \*/ BOOL fCut,

/\* [in] \*/ BOOL fMoved)

{

// we save if something fails but we are trying to do as much as possible

HRESULT hrRet = S\_OK; // hr to return

HRESULT hr = S\_OK;

/+

CComPtr<IVsUIHierarchyWindow> srpIVsUIHierarchyWindow;

hr = \_VxModule.GetIVsUIHierarchyWindow(

/\* REFGUID rguidPersistenceSlot \*/GUID\_SolutionExplorer,

/\*IVsUIHierarchyWindow \*\*ppIVsUIHierarchyWindow\*/ &srpIVsUIHierarchyWindow);

if (FAILED(hr))

hrRet = hr;

if (!srpIVsUIHierarchyWindow)

hrRet = E\_UNEXPECTED;

+/

for (ULONG i = 0; i < mItemSelDragged.length; i++)

{

if((fMoved && fDropped) || fCut)

{

CFileNode pFileNode = cast(CFileNode) VSITEMID2Node(mItemSelDragged[i].itemid);

if (!pFileNode)

{

CHierContainer pFolderNode = cast(CHierContainer) VSITEMID2Node(mItemSelDragged[i].itemid);

if(pFolderNode)

if(auto parent = pFolderNode.GetParent())

hr = parent.Delete(pFolderNode, this);

continue;

}

bool fOpen = FALSE;

bool fDirty = FALSE;

bool fOpenByUs = FALSE;

hr = pFileNode.GetDocInfo(

/\* [out, opt] BOOL\* pfOpen \*/ &fOpen, // true if the doc is opened

/\* [out, opt] BOOL\* pfDirty \*/ &fDirty, // true if the doc is dirty

/\* [out, opt] BOOL\* pfOpenByUs \*/ &fOpenByUs, // true if opened by our project

/\* [out, opt] VSDOCCOOKIE\* pVsDocCookie\*/ null);// VSDOCCOOKIE if open

if (FAILED(hr))

continue;

// do not close it if the doc is dirty or we do not own it

if (fDirty || (fOpen && !fOpenByUs))

continue;

// close it if opened

if (fOpen)

{

hr = pFileNode.CloseDoc(SLNSAVEOPT\_NoSave);

if (FAILED(hr))

hrRet = hr;

}

BOOL res;

hr = RemoveItem(0, mItemSelDragged[i].itemid, &res);

if (FAILED(hr))

hrRet = hr;

}

else

{

/+

if (srpIVsUIHierarchyWindow)

hr = srpIVsUIHierarchyWindow->ExpandItem(QI\_cast<IVsUIHierarchy>(this), m\_pItemSelDragged[i].itemid, EXPF\_UnCutHighlightItem);

if (FAILED(hr))

hrRet = hr;

+/

}

}

mItemSelDragged.length = 0;

return hrRet;

}

///////////////////////////////////////////////////////////////////////

void ClearLineChanges()

{

auto langsvc = Package.GetLanguageService();

searchNode(GetRootNode(), delegate (CHierNode n) {

string file = n.GetCanonicalName();

if(auto src = langsvc.GetSource(file))

src.ClearLineChanges();

return false;

});

}

//////////////////////////////////////////////////////////////

dte.ConfigurationManager getConfigurationManager()

{

dte.ConfigurationManager mgr;

if(IVsExtensibility3 ext = queryService!(dte.IVsExtensibility, IVsExtensibility3))

{

IUnknown obj;

if(ext.GetConfigMgr(this, VSITEMID\_ROOT, &obj) == S\_OK)

{

if (obj.QueryInterface(&dte.ConfigurationManager.iid, cast(void\*\*) &mgr) == S\_OK)

assert(mgr);

obj.Release();

}

ext.Release();

}

return mgr;

}

static xml.Document readXML(string fileName)

{

try

{

string text = cast(string) read(fileName);

size\_t decidx = 0;

if(decode(text, decidx) == 0xfeff)

text = text[decidx..$];

if(!startsWith(text, "<?xml"))

text = `<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>` ~ text;

xml.Document doc = xml.readDocument(text);

return doc;

}

catch(xml.RecodeException rc)

{

string msg = rc.toString();

writeToBuildOutputPane(msg);

logCall(msg);

}

catch(xml.XmlException rc)

{

string msg = rc.toString();

writeToBuildOutputPane(msg);

logCall(msg);

}

return null;

}

bool parseXML()

{

string fileName;

try

{

fileName = toUTF8(mFilename);

mDoc = readXML(fileName);

if(!mDoc)

goto fail;

xml.Element root = xml.getRoot(mDoc);

if(xml.Element el = xml.getElement(root, "ProjectGuid"))

mProjectGUID = uuid(el.text());

string projectName = getNameWithoutExt(fileName);

CProjectNode rootnode = newCom!CProjectNode(fileName, this);

xml.Element[] propItems = xml.elementsById(root, "Folder");

foreach(item; propItems)

{

projectName = xml.getAttribute(item, "name");

parseContainer(rootnode, item);

}

rootnode.SetName(projectName);

xml.Element[] cfgItems = xml.elementsById(root, "Config");

foreach(cfg; cfgItems)

{

string name = xml.getAttribute(cfg, "name");

string platform = xml.getAttribute(cfg, "platform");

if(platform.length == 0)

platform = kPlatforms[0];

Config config = mConfigProvider.addConfig(name, platform);

config.GetProjectOptions().readXML(cfg);

}

SetRootNode(rootnode);

return true;

}

catch(Exception e)

{

writeToBuildOutputPane(e.toString());

logCall(e.toString());

}

fail:

string projectName = getNameWithoutExt(fileName);

CProjectNode rootnode = newCom!CProjectNode("", this);

rootnode.SetName("Failed to load " ~ projectName);

SetRootNode(rootnode);

return false;

}

void parseContainer(CHierContainer cont, xml.Element item)

{

xml.Element[] folderItems = xml.elementsById(item, "Folder");

foreach(folder; folderItems)

{

string name = xml.getAttribute(folder, "name");

CHierContainer node = newCom!CFolderNode(name);

cont.Add(node);

parseContainer(node, folder);

}

xml.Element[] fileItems = xml.elementsById(item, "File");

foreach(file; fileItems)

{

string fileName = xml.getAttribute(file, "path");

CFileNode node = newCom!CFileNode(fileName);

static parseFileOptions(CFileNode node, string cfg, xml.Element file)

{

node.SetTool(cfg, xml.getAttribute(file, "tool"));

node.SetDependencies(cfg, xml.getAttribute(file, "dependencies"));

node.SetOutFile(cfg, xml.getAttribute(file, "outfile"));

node.SetCustomCmd(cfg, xml.getAttribute(file, "customcmd"));

node.SetAdditionalOptions(cfg, xml.getAttribute(file, "addopt"));

node.SetLinkOutput(cfg, xml.getAttribute(file, "linkoutput") == "true");

node.SetUptodateWithSameTime(cfg, xml.getAttribute(file, "uptodateWithSameTime") == "true");

}

parseFileOptions(node, null, file);

node.SetPerConfigOptions(xml.getAttribute(file, "perConfig") == "true");

xml.Element[] cfgItems = xml.elementsById(file, "Config");

foreach(cfgItem; cfgItems)

{

string cfg = xml.getAttribute(cfgItem, "name");

parseFileOptions(node, cfg, cfgItem);

}

cont.Add(node);

}

}

static bool saveXML(xml.Document doc, string filename)

{

try

{

string[] result = xml.writeDocument(doc);

string output;

foreach(ostr; result)

output ~= ostr ~ "\n";

std.file.write(filename, output);

return true;

}

catch(Exception e)

{

string msg = e.toString();

writeToBuildOutputPane(msg);

logCall(msg);

}

return false;

}

xml.Document createDoc()

{

xml.Document doc = xml.newDocument("DProject");

xml.Element root = xml.getRoot(doc);

root ~= new xml.Element("ProjectGuid", GUID2string(mProjectGUID));

mConfigProvider.addConfigsToXml(doc);

createDocHierarchy(root, GetProjectNode());

return doc;

}

static void createDocHierarchy(xml.Element elem, CHierContainer container)

{

auto xmlcontainer = new xml.Element("Folder");

xml.setAttribute(xmlcontainer, "name", container.GetName());

for(CHierNode node = container.GetHeadEx(false); node; node = node.GetNext(false))

{

if(CHierContainer cont = cast(CHierContainer) node)

createDocHierarchy(xmlcontainer, cont);

else if(CFileNode file = cast(CFileNode) node)

{

auto xmlfile = new xml.Element("File");

xml.setAttribute(xmlfile, "path", file.GetFilename());

if(file.GetPerConfigOptions())

xml.setAttribute(xmlfile, "perConfig", "true");

static void setAttrIfNotEmpty(xml.Element xmlFile, string attr, string val)

{

if(val.length)

xml.setAttribute(xmlFile, attr, val);

}

static void writeFileConfig(xml.Element xmlFile, CFileNode file, string cfg)

{

setAttrIfNotEmpty(xmlFile, "tool", file.GetTool(cfg));

setAttrIfNotEmpty(xmlFile, "dependencies", file.GetDependencies(cfg));

setAttrIfNotEmpty(xmlFile, "outfile", file.GetOutFile(cfg));

setAttrIfNotEmpty(xmlFile, "customcmd", file.GetCustomCmd(cfg));

setAttrIfNotEmpty(xmlFile, "addopt", file.GetAdditionalOptions(cfg));

if(file.GetLinkOutput(cfg))

xml.setAttribute(xmlFile, "linkoutput", "true");

if(file.GetUptodateWithSameTime(cfg))

xml.setAttribute(xmlFile, "uptodateWithSameTime", "true");

}

writeFileConfig(xmlfile, file, null);

auto cfgs = file.GetConfigOptions().keys;

foreach(cfg; cfgs)

{

auto xmlcfg = new xml.Element("Config");

xml.setAttribute(xmlcfg, "name", cfg);

writeFileConfig(xmlcfg, file, cfg);

xmlfile ~= xmlcfg;

}

xmlcontainer ~= xmlfile;

}

}

elem ~= xmlcontainer;

}

string GetFilename() { return mFilename; }

string GetName() { return mName; }

string GetCaption() { return mCaption; }

void SetCaption(string caption) { mCaption = caption; }

private:

ProjectFactory mFactory;

string mName;

string mFilename;

string mEditLabel;

string mCaption;

GUID mProjectGUID;

ConfigProvider mConfigProvider;

ExtProject mExtProject;

bool mfDragSource;

DropDataType mDDT;

VSITEMSELECTION[] mItemSelDragged;

xml.Document mDoc;

}

void checkDustMiteDirs(string dustmitepath)

{

if (std.file.exists(dustmitepath) && !std.file.dirEntries(dustmitepath, SpanMode.shallow).empty())

{

string msg = "Folder " ~ dustmitepath ~ " already exists and is not empty. Remove to continue?";

int msgRet = UtilMessageBox(msg, MB\_OKCANCEL | MB\_ICONEXCLAMATION, "DustMite");

if (msgRet != IDOK)

throw new Exception("DustMite operation cancelled");

try

{

rmdirRecurse(dustmitepath);

}

catch(Exception e)

{

// ok to swallow exception if directory is left empty

if (!std.file.dirEntries(dustmitepath, SpanMode.shallow).empty())

throw e;

}

}

string reducedpath = dustmitepath ~ ".reduced";

if (std.file.exists(reducedpath))

{

string msg = "Folder " ~ reducedpath ~ " already exists. Remove to continue?";

int msgRet = UtilMessageBox(msg, MB\_OKCANCEL | MB\_ICONEXCLAMATION, "DustMite");

if (msgRet != IDOK)

throw new Exception("DustMite operation cancelled");

rmdirRecurse(reducedpath);

}

}

string getSelectedTextInBuildPane()

{

if(auto opane = getBuildOutputPane())

{

scope(exit) release(opane);

if(auto owin = qi\_cast!IVsTextView(opane))

{

BSTR selText;

if (owin.GetSelectedText (&selText) == S\_OK)

return detachBSTR(selText);

}

}

return null;

}

string getCurrentErrorText()

{

if (auto tasklist = queryService!(SVsErrorList, IVsTaskList2)())

{

scope(exit) release(tasklist);

IVsTaskItem item;

if (tasklist.GetCaretPos(&item) == S\_OK && item)

{

scope(exit) release(item);

BSTR text;

if (item.get\_Text (&text) == S\_OK)

return detachBSTR(text);

}

}

return null;

}

HRESULT DustMiteProject()

{

auto solutionBuildManager = queryService!(IVsSolutionBuildManager)();

scope(exit) release(solutionBuildManager);

IVsHierarchy phier;

if(solutionBuildManager.get\_StartupProject(&phier) != S\_OK)

return E\_FAIL;

Project proj = qi\_cast!Project(phier);

scope(exit) release(phier);

Config cfg;

IVsProjectCfg activeCfg;

scope(exit) release(activeCfg);

if(solutionBuildManager && proj)

if(solutionBuildManager.FindActiveProjectCfg(null, null, proj, &activeCfg) == S\_OK)

cfg = qi\_cast!Config(activeCfg);

if(!cfg)

return E\_FAIL;

string errmsg = getSelectedTextInBuildPane();

if (errmsg.length == 0)

errmsg = getCurrentErrorText();

if (errmsg.length == 0)

errmsg = "Internal error";

string projname = proj.GetCaption();

if (projname.length == 0)

projname = proj.GetName();

if (projname.length == 0)

projname = baseName(proj.GetFilename());

string msg = format("Do you want to reduce project %s for error message \"%s\"?\n" ~

"Visual D will try to create a clean copy of your project, but\n" ~

"you might also consider making a backup of the project!", projname, errmsg);

string caption = "DustMite";

int msgRet = UtilMessageBox(msg, MB\_YESNO | MB\_ICONEXCLAMATION, caption);

if (msgRet != IDYES)

return S\_FALSE;

string workdir = cfg.GetProjectDir();

auto pane = getVisualDOutputPane();

scope(exit) release(pane);

clearOutputPane();

if(!pane)

return S\_FALSE;

pane.Activate();

string npath, nworkdir, cmdline, cmdfile, dustfile, cmd;

try

{

string commonpath = commonProjectFolder(proj);

string dustmitepath = buildPath(dirName(commonpath), baseName(commonpath) ~ ".dustmite"); // need to strip trailing '\'

checkDustMiteDirs(dustmitepath);

npath = copyProjectFolder(proj, dustmitepath);

if (npath.length == 0)

return pane.OutputString("cannot determine common root folder for all sources\n"w.ptr), S\_FALSE;

pane.OutputString(\_toUTF16z("created clean copy of the project in " ~ dustmitepath ~ "\n"));

nworkdir = npath; // TODO

string nintdir = makeFilenameAbsolute(cfg.GetIntermediateDir(), nworkdir);

string noutdir = makeFilenameAbsolute(cfg.GetOutDir(), nworkdir);

mkdirRecurse(nworkdir);

mkdirRecurse(nintdir);

mkdirRecurse(noutdir);

std.file.write(normalizeDir(nworkdir) ~ "empty.txt", ""); // dustmite needs non-empty directories

std.file.write(normalizeDir(nintdir) ~ "empty.txt", "");

std.file.write(normalizeDir(noutdir) ~ "empty.txt", "");

if (nworkdir != npath)

cmdline ~= "cd " ~ quoteFilename(makeRelative(nworkdir, npath));

cmdline ~= cfg.getCommandLine();

cmdfile = npath ~ "build.dustmite.bat";

std.file.write(cmdfile, cmdline);

cmdfile = makeRelative(cmdfile, npath);

string dustcmd = quoteFilename(cmdfile) ~ " | find \"" ~ errmsg ~ "\"";

dustcmd = dustcmd.replace("\"", "\\\"");

string intdir = makeFilenameAbsolute(cfg.GetIntermediateDir(), workdir);

mkdirRecurse(intdir);

dustfile = intdir ~ "[\\dustmite.cmd](file:///\\dustmite.cmd)";

string opts = "--strip-comments --split \*.bat:lines";

cmd = Package.GetGlobalOptions().findDmdBinDir() ~ "dustmite " ~ opts ~ " " ~ quoteFilename(npath[0..$-1]) ~ " \"" ~ dustcmd ~ "\"";

std.file.write(dustfile, cmd ~ "\npause\n");

std.process.spawnShell(quoteFilename(dustfile), null, std.process.Config.none, nworkdir);

pane.OutputString(\_toUTF16z("Spawned dustmite, check new console window for output...\n"));

}

catch(Exception e)

{

pane.OutputString(\_toUTF16z(e.msg ~ "\n"));

return S\_FALSE;

}

return S\_OK;

}

class DustMiteThread : CBuilderThread

{

this(Config cfg, string buildDir)

{

super(cfg);

mBuildDir = buildDir;

}

override string GetBuildDir()

{

return mBuildDir;

}

override bool needsOutputParser() { return false; }

string mBuildDir;

}

// This file is part of Visual D

//

// Visual D integrates the D programming language into Visual Studio

// Copyright (c) 2010 by Rainer Schuetze, All Rights Reserved

//

// Distributed under the Boost Software License, Version 1.0.

// See accompanying file LICENSE\_1\_0.txt or copy at <http://www.boost.org/LICENSE_1_0.txt>

module visuald.expansionprovider;

import visuald.windows;

import std.ascii;

import std.string;

import std.utf;

import visuald.comutil;

import visuald.logutil;

import visuald.hierutil;

import visuald.dpackage;

import visuald.pkgutil;

import visuald.dlangsvc;

import vdc.lexer;

import sdk.vsi.textmgr;

import sdk.vsi.textmgr2;

import sdk.vsi.vsshell;

import sdk.vsi.singlefileeditor;

import sdk.win32.xmldom;

///////////////////////////////////////////////////////////////////////////////

struct DefaultFieldValue

{

string field;

string value;

}

bool ContainsExclusive(ref TextSpan span, int line, int col)

{

if (line > span.iStartLine && line < span.iEndLine)

return true;

if (line == span.iStartLine)

return (col > span.iStartIndex && (line < span.iEndLine ||

(line == span.iEndLine && col < span.iEndIndex)));

if (line == span.iEndLine)

return col < span.iEndIndex;

return false;

}

class ExpansionProvider : DisposingComObject, IVsExpansionClient

{

IVsTextView mView;

Source mSource;

IVsExpansion vsExpansion;

IVsExpansionSession expansionSession;

bool expansionActive;

bool expansionPrepared;

bool completorActiveDuringPreExec;

DefaultFieldValue[] fieldDefaults; // CDefaultFieldValues

string titleToInsert;

string pathToInsert;

this(Source src)

{

mSource = src;

vsExpansion = qi\_cast!(IVsExpansion)(src.GetTextLines());

assert(vsExpansion);

}

override void Dispose()

{

EndTemplateEditing(true);

mSource = null;

vsExpansion = release(vsExpansion);

mView = release(mView);

}

override HRESULT QueryInterface(in IID\* riid, void\*\* pvObject)

{

if(queryInterface!(IVsExpansionClient) (this, riid, pvObject))

return S\_OK;

return super.QueryInterface(riid, pvObject);

}

bool HandleQueryStatus(ref GUID guidCmdGroup, uint nCmdId, out int hr)

{

// in case there's something to conditinally support later on...

hr = 0;

return false;

}

bool GetExpansionSpan(TextSpan \*span)

{

assert(expansionSession);

int hr = expansionSession.GetSnippetSpan(span);

return SUCCEEDED(hr);

}

bool HandlePreExec(in GUID\* guidCmdGroup, uint nCmdId, uint nCmdexecopt, in VARIANT\* pvaIn, VARIANT\* pvaOut)

{

if(!expansionActive || !expansionSession)

return false;

completorActiveDuringPreExec = IsCompletorActive(mView);

if(\*guidCmdGroup == CMDSETID\_StandardCommandSet2K)

{

switch (nCmdId) {

case ECMD\_CANCEL:

if(completorActiveDuringPreExec)

return false;

EndTemplateEditing(true);

return true;

case ECMD\_RETURN:

bool leaveCaret = false;

int line = 0, col = 0;

if(SUCCEEDED(mView.GetCaretPos(&line, &col)))

{

TextSpan span;

if(GetExpansionSpan(&span))

if(!ContainsExclusive(span, line, col))

leaveCaret = true;

}

if(completorActiveDuringPreExec)

return false;

EndTemplateEditing(leaveCaret);

if(leaveCaret)

return false;

return true;

case ECMD\_BACKTAB:

if(completorActiveDuringPreExec)

return false;

expansionSession.GoToPreviousExpansionField();

return true;

case ECMD\_TAB:

if(completorActiveDuringPreExec)

return false;

expansionSession.GoToNextExpansionField(0); // fCommitIfLast=false

return true;

default:

break;

}

}

return false;

}

bool HandlePostExec(in GUID\* guidCmdGroup, uint nCmdId, uint nCmdexecopt, bool commit, in VARIANT\* pvaIn, VARIANT\* pvaOut)

{

if(\*guidCmdGroup == CMDSETID\_StandardCommandSet2K)

{

switch (nCmdId) {

case ECMD\_RETURN:

if (completorActiveDuringPreExec && commit) {

// if the completor was active during the pre-exec we want to let it handle the command first

// so we didn't deal with this in pre-exec. If we now get the command, we want to end

// the editing of the expansion. We also return that we handled the command so auto-indenting doesn't happen

EndTemplateEditing(false);

completorActiveDuringPreExec = false;

return true;

}

break;

default:

break;

}

}

completorActiveDuringPreExec = false;

return false;

}

bool DisplayExpansionBrowser(IVsTextView view, string prompt, string[] types, bool includeNullType,

string[] kinds, bool includeNullKind)

{

if (expansionActive)

EndTemplateEditing(true);

if (mSource.IsCompletorActive())

mSource.DismissCompletor();

mView = view;

IVsTextManager2 textmgr = queryService!(VsTextManager, IVsTextManager2);

if(!textmgr)

return false;

scope(exit) release(textmgr);

IVsExpansionManager exmgr;

textmgr.GetExpansionManager(&exmgr);

if (!exmgr)

return false;

scope(exit) release(exmgr);

BSTR[] bstrTypes;

foreach(type; types)

bstrTypes ~= allocBSTR(type);

BSTR[] bstrKinds;

foreach(kind; kinds)

bstrKinds ~= allocBSTR(kind);

auto bstrPrompt = ScopedBSTR(prompt);

int hr = exmgr.InvokeInsertionUI(mView, // pView

this, // pClient

g\_languageCLSID, // guidLang

bstrTypes.ptr, // bstrTypes

bstrTypes.length, // iCountTypes

includeNullType ? 1 : 0, // fIncludeNULLType

bstrKinds.ptr, // bstrKinds

bstrKinds.length, // iCountKinds

includeNullKind ? 1 : 0, // fIncludeNULLKind

bstrPrompt, // bstrPrefixText

">"); //bstrCompletionChar

foreach(type; bstrTypes)

freeBSTR(type);

foreach(kind; bstrKinds)

freeBSTR(kind);

return SUCCEEDED(hr);

}

bool InsertSpecificExpansion(IVsTextView view, IXMLDOMNode snippet, TextSpan pos, string relativePath)

{

if (expansionActive)

EndTemplateEditing(true);

if (mSource.IsCompletorActive())

mSource.DismissCompletor();

mView = view;

auto bstrRelPath = ScopedBSTR(relativePath);

int hr = vsExpansion.InsertSpecificExpansion(snippet, pos, this, g\_languageCLSID, bstrRelPath, &expansionSession);

if (hr != S\_OK || !expansionSession)

EndTemplateEditing(true);

else

{

// When inserting a snippet it is possible that the edit session is ended inside the insert

// function (e.g. if the template has no editable fields). In this case we should not stay

// in template edit mode because otherwise our filter will stole messages to the editor.

if (!expansionActive) {

expansionSession = null;

}

return true;

}

return false;

}

bool IsCompletorActive(IVsTextView view)

{

if (mSource.IsCompletorActive())

return true;

IVsTextViewEx viewex = qi\_cast!(IVsTextViewEx)(view);

scope(exit) release(viewex);

if (viewex)

return viewex.IsCompletorWindowActive() == S\_OK;

return false;

}

bool InsertNamedExpansion(IVsTextView view, BSTR title, BSTR path, TextSpan pos, bool showDisambiguationUI)

{

if (mSource.IsCompletorActive())

mSource.DismissCompletor();

mView = view;

if (expansionActive)

EndTemplateEditing(true);

int hr = vsExpansion.InsertNamedExpansion(title, path, pos, this,

g\_languageCLSID, showDisambiguationUI ? 1 : 0, &expansionSession);

if (hr != S\_OK || !expansionSession)

{

EndTemplateEditing(true);

return false;

}

if (hr == S\_OK)

{

// When inserting a snippet it is possible that the edit session is ended inside the insert

// function (e.g. if the template has no editable fields). In this case we should not stay

// in template edit mode because otherwise our filter will stole messages to the editor.

if (!expansionActive)

expansionSession = null;

return true;

}

return false;

}

/// Returns S\_OK if match found, S\_FALSE if expansion UI is shown, and error otherwise

int InvokeExpansionByShortcut(IVsTextView view, wstring shortcut, ref TextSpan span, bool showDisambiguationUI, out string title, out string path)

{

if (expansionActive)

EndTemplateEditing(true);

mView = view;

title = "";

path = "";

mView = view;

IVsTextManager2 textmgr = queryService!(VsTextManager, IVsTextManager2);

if(!textmgr)

return E\_FAIL;

scope(exit) release(textmgr);

IVsExpansionManager exmgr;

textmgr.GetExpansionManager(&exmgr);

if (!exmgr)

return E\_FAIL;

scope(exit) release(exmgr);

BSTR bstrPath, bstrTitle;

int hr = exmgr.GetExpansionByShortcut(this, g\_languageCLSID, \_toUTF16zw(shortcut), mView,

&span, showDisambiguationUI ? 1 : 0, &bstrPath, &bstrTitle);

if(FAILED(hr) || !bstrPath || !bstrTitle)

return S\_FALSE; // when no shortcut found, do nothing

if(!InsertNamedExpansion(view, bstrTitle, bstrPath, span, showDisambiguationUI))

hr = E\_FAIL;

path = detachBSTR(bstrPath);

title = detachBSTR(bstrTitle);

return hr;

}

// for an example of GetExpansionFunction, see

// <http://msdn.microsoft.com/en-us/library/microsoft.visualstudio.package.expansionfunction%28VS.80%29.aspx>

IVsExpansionFunction GetExpansionFunction(string func, string fieldName)

{

string functionName;

string[] rgFuncParams;

if (func.length == 0)

return null;

bool inIdent = false;

bool inParams = false;

int token = 0;

// initialize the vars needed for our super-complex function parser :-)

for (int i = 0, n = func.length; i < n; i++)

{

char ch = func[i];

// ignore and skip whitespace

if (!isWhite(ch))

{

switch (ch)

{

case ',':

if (!inIdent || !inParams)

i = n; // terminate loop

else

{

// we've hit a comma, so end this param and move on...

string name = func[token .. i];

rgFuncParams ~= name;

inIdent = false;

}

break;

case '(':

if (!inIdent || inParams)

i = n; // terminate loop

else

{

// we've hit the (, so we know the token before this is the name of the function

functionName = func[token .. i];

inIdent = false;

inParams = true;

}

break;

case ')':

if (!inParams)

i = n; // terminate loop

else

{

if (inIdent)

{

// save last param and stop

string name = func[token .. i];

rgFuncParams ~= name;

inIdent = false;

}

i = n; // terminate loop

}

break;

default:

if (!inIdent)

{

inIdent = true;

token = i;

}

break;

}

}

}

if(functionName.length > 0)

{

if(ExpansionFunction expfunc = CreateExpansionFunction(functionName))

{

expfunc.fieldName = fieldName;

expfunc.args = rgFuncParams;

return expfunc;

}

}

return null;

}

ExpansionFunction CreateExpansionFunction(string functionName)

{

return newCom!ExpansionFunction(this);

}

void PrepareTemplate(string title, string path)

{

assert(title.length);

// stash the title and path for when we actually insert the template

titleToInsert = title;

pathToInsert = path;

expansionPrepared = true;

}

void SetFieldDefault(string field, string value)

{

assert(expansionPrepared);

//assert(field && value);

// we have an expansion "prepared" to insert, so we can now save this

// field default to set when the expansion is actually inserted

fieldDefaults ~= DefaultFieldValue(field, value);

}

void BeginTemplateEditing(int line, int col)

{

assert(expansionPrepared);

TextSpan tsInsert;

tsInsert.iStartLine = tsInsert.iEndLine = line;

tsInsert.iStartIndex = tsInsert.iEndIndex = col;

auto bstrTitle = ScopedBSTR(titleToInsert);

auto bstrPath = ScopedBSTR(pathToInsert);

int hr = vsExpansion.InsertNamedExpansion(bstrTitle, bstrPath, tsInsert,

this, g\_languageCLSID, 0, &expansionSession);

if (hr != S\_OK)

EndTemplateEditing(true);

pathToInsert = null;

titleToInsert = null;

}

void EndTemplateEditing(bool leaveCaret)

{

if (!expansionActive || !expansionSession)

{

expansionActive = false;

return;

}

expansionSession.EndCurrentExpansion(leaveCaret ? 1 : 0); // fLeaveCaret=true

expansionSession = null;

expansionActive = false;

}

bool GetFieldSpan(string field, TextSpan\* pts)

{

assert(expansionSession);

if (!expansionSession)

return false;

auto bstrField = ScopedBSTR(field);

expansionSession.GetFieldSpan(bstrField, pts);

return true;

}

bool GetFieldValue(string field, out string value)

{

assert(expansionSession);

if (!expansionSession)

return false;

BSTR bstrValue;

auto bstrField = ScopedBSTR(field);

int hr = expansionSession.GetFieldValue(bstrField, &bstrValue);

value = detachBSTR(bstrValue);

return hr == S\_OK;

}

override int EndExpansion()

{

mixin(LogCallMix);

expansionActive = false;

expansionSession = null;

return S\_OK;

}

override int FormatSpan(IVsTextLines buffer, in TextSpan\* ts)

{

mixin(LogCallMix);

assert(mSource.GetTextLines() is buffer);

int rc = E\_NOTIMPL;

if (mSource.EnableFormatSelection())

{

// We should not merge edits in this case because it might clobber the

// $varname$ spans which are markers for yellow boxes.

// using (EditArray edits = new EditArray(mSource, mView, false, SR.GetString(SR.FormatSpan))) {

// mSource.ReformatSpan(edits, span);

// edits.ApplyEdits();

//}

rc = mSource.ReindentLines(mView, ts.iStartLine, ts.iEndLine);

}

return rc;

}

override int IsValidKind(IVsTextLines buffer, in TextSpan \*ts, in BSTR bstrKind, BOOL \*fIsValid)

{

mixin(LogCallMix);

\*fIsValid = 0;

assert(mSource.GetTextLines() is buffer);

\*fIsValid = 1;

return S\_OK;

}

override int IsValidType(IVsTextLines buffer, in TextSpan\* ts, in BSTR\* rgTypes, in int iCountTypes, BOOL \*fIsValid)

{

mixin(LogCallMix);

\*fIsValid = 0;

assert(mSource.GetTextLines() is buffer);

\*fIsValid = 1;

return S\_OK;

}

override int OnItemChosen(in BSTR pszTitle, in BSTR pszPath)

{

mixin(LogCallMix2);

TextSpan ts;

mView.GetCaretPos(&ts.iStartLine, &ts.iStartIndex);

ts.iEndLine = ts.iStartLine;

ts.iEndIndex = ts.iStartIndex;

if (expansionSession) // previous session should have been ended by now!

EndTemplateEditing(true);

// insert the expansion

// TODO: Replace the last parameter with the right string to display as a name of undo operation

// CompoundActionBase cab = CompoundActionFactory.GetCompoundAction(mView, mSource, SR.FormatSpan));

return vsExpansion.InsertNamedExpansion(pszTitle, pszPath, // Bug: VSCORE gives us unexpanded path

ts, this, g\_languageCLSID, 0, // fShowDisambiguationUI, (FALSE)

&expansionSession);

}

override int PositionCaretForEditing(IVsTextLines pBuffer, in TextSpan\* ts)

{

mixin(LogCallMix2);

// NOP

return S\_OK;

}

override int OnAfterInsertion(IVsExpansionSession session)

{

mixin(LogCallMix);

return S\_OK;

}

override int OnBeforeInsertion(IVsExpansionSession session)

{

mixin(LogCallMix);

if (!session)

return E\_UNEXPECTED;

expansionPrepared = false;

expansionActive = true;

// stash the expansion session pointer while the expansion is active

if (!expansionSession)

expansionSession = session;

else

// these better be the same!

assert(expansionSession is session);

// now set any field defaults that we have.

foreach (ref DefaultFieldValue dv; fieldDefaults)

{

auto bstrField = ScopedBSTR(dv.field);

auto bstrValue = ScopedBSTR(dv.value);

expansionSession.SetFieldDefault(bstrField, bstrValue);

}

fieldDefaults.length = 0;

return S\_OK;

}

override int GetExpansionFunction(IXMLDOMNode xmlFunctionNode, in BSTR bstrFieldName, IVsExpansionFunction\* func)

{

//mixin(LogCallMix);

BSTR text;

if(int hr = xmlFunctionNode.get\_text(&text))

return hr;

string innerText = detachBSTR(text);

\*func = GetExpansionFunction(innerText, to\_string(bstrFieldName));

return S\_OK;

}

}

class ExpansionFunction : DComObject, IVsExpansionFunction

{

ExpansionProvider mProvider;

string fieldName;

string[] args;

string[] list;

this(ExpansionProvider provider)

{

mProvider = addref(provider);

}

~this()

{

mProvider = release(mProvider);

}

override HRESULT QueryInterface(in IID\* riid, void\*\* pvObject)

{

if(queryInterface!(IVsExpansionFunction) (this, riid, pvObject))

return S\_OK;

return super.QueryInterface(riid, pvObject);

}

/+

/// <include file='doc\ExpansionProvider.uex' path='docs/doc[@for="ExpansionFunction.GetCurrentValue"]/\*' />

public abstract string GetCurrentValue();

+/

/// <summary>Override this method if you want intellisense drop support on a list of possible values.</summary>

string[] GetIntellisenseList()

{

return null;

}

/+

/// Gets the value of the specified argument, resolving any fields referenced in the argument.

/// In the substitution, "$$" is replaced with "$" and any floating '$' signs are left unchanged,

/// for example "$US 23.45" is returned as is. Only if the two dollar signs enclose a string of

/// letters or digits is this considered a field name (e.g. "$foo123$"). If the field is not found

/// then the unresolved string "$foo" is returned.

string GetArgument(int index)

{

if (index < 0 || index >= args.length)

return null;

string arg = args[index];

if (arg.length == 0)

return null;

int i = indexOf(arg, '$');

if (i >= 0)

{

int j = arg[

StringBuilder sb = new StringBuilder();

int len = arg.length;

int start = 0;

while (i >= 0 && i + 1 < len)

{

sb.Append(arg.Substring(start, i - start));

start = i;

i++;

if (arg[i] == '$') {

sb.Append('$');

start = i + 1; // $$ is resolved to $.

} else {

// parse name of variable.

int j = i;

for (; j < len; j++) {

if (!Char.IsLetterOrDigit(arg[j]))

break;

}

if (j == len) {

// terminating '$' not found.

sb.Append('$');

start = i;

break;

} else if (arg[j] == '$') {

string name = arg.Substring(i, j - i);

string value;

if (GetFieldValue(name, out value)) {

sb.Append(value);

} else {

// just return the unresolved variable.

sb.Append('$');

sb.Append(name);

sb.Append('$');

}

start = j + 1;

} else {

// invalid syntax, e.g. "$US 23.45" or some such thing

sb.Append('$');

sb.Append(arg.Substring(i, j - i));

start = j;

}

}

i = arg.IndexOf('$', start);

}

if (start < len) {

sb.Append(arg.Substring(start, len - start));

}

arg = sb.ToString();

}

// remove quotes around string literals.

if (arg.Length > 2 && arg[0] == '"' && arg[arg.Length - 1] == '"') {

arg = arg.Substring(1, arg.Length - 2);

} else if (arg.Length > 2 && arg[0] == '\'' && arg[arg.Length - 1] == '\'') {

arg = arg.Substring(1, arg.Length - 2);

}

return arg;

}

+/

bool GetFieldValue(string name, out string value)

{

if (mProvider && mProvider.expansionSession)

{

auto fieldName = ScopedBSTR(name);

BSTR fieldValue;

int hr = mProvider.expansionSession.GetFieldValue(fieldName, &fieldValue);

value = detachBSTR(fieldValue);

return SUCCEEDED(hr);

}

return false;

}

public TextSpan GetSelection()

{

TextSpan result;

if (mProvider && mProvider.mView)

{

int hr = GetSelectionForward(mProvider.mView, &result.iStartLine, &result.iStartIndex, &result.iEndLine, &result.iEndIndex);

assert(SUCCEEDED(hr));

}

return result;

}

override int FieldChanged(in BSTR bstrField, BOOL \*fRequeryValue)

{

// Returns true if we care about this field changing.

// We care if the field changes if one of the arguments refers to it.

if (args.length)

{

string var = "$" ~ to\_string(bstrField) ~ "$";

foreach (string arg; args)

{

if (arg == var)

{

\*fRequeryValue = 1; // we care!

return S\_OK;

}

}

}

\*fRequeryValue = 0;

return S\_OK;

}

override HRESULT GetDefaultValue(/+[out]+/BSTR \*bstrValue, /+[out]+/ BOOL \*fHasDefaultValue)

{

// This must call GetCurrentValue since during initialization of the snippet

// VS will call GetDefaultValue and not GetCurrentValue.

return GetCurrentValue(bstrValue, fHasDefaultValue);

}

override HRESULT GetCurrentValue(/+[out]+/BSTR \*bstrValue, /+[out]+/ BOOL \*fHasDefaultValue)

{

\*bstrValue = allocBSTR(""); // \_toUTF16z("");

\*fHasDefaultValue = !bstrValue ? 0 : 1;

return S\_OK;

}

override int GetFunctionType(DWORD\* pFuncType)

{

if (!list.length)

list = GetIntellisenseList();

\*pFuncType = list.length ? eft\_List : eft\_Value;

return S\_OK;

}

override int GetListCount(int\* iListCount)

{

if (!list.length)

list = GetIntellisenseList();

\*iListCount = list.length;

return S\_OK;

}

override int GetListText(in int iIndex, BSTR\* ppszText)

{

if (!list.length)

list = GetIntellisenseList();

if (iIndex < list.length)

\*ppszText = allocBSTR(list[iIndex]);

else

\*ppszText = null;

return S\_OK;

}

override int ReleaseFunction()

{

mProvider = release(mProvider);

return S\_OK;

}

/+

// todo: for some reason VsExpansionManager is wrong.

[Guid("4970C2BC-AF33-4a73-A34F-18B0584C40E4")]

internal class SVsExpansionManager {

}

+/

}

// This file is part of Visual D

//

// Visual D integrates the D programming language into Visual Studio

// Copyright (c) 2010 by Rainer Schuetze, All Rights Reserved

//

// Distributed under the Boost Software License, Version 1.0.

// See accompanying file LICENSE\_1\_0.txt or copy at <http://www.boost.org/LICENSE_1_0.txt>

module visuald.fileutil;

import sdk.port.base;

import sdk.win32.shellapi;

import stdext.array;

import stdext.file;

import stdext.string;

import stdext.path;

import std.algorithm;

import std.path;

import std.file;

import std.string;

import std.conv;

import std.utf;

import std.stdio;

import std.regex;

//-----------------------------------------------------------------------------

long[string] gCachedFileTimes;

alias AssociativeArray!(string, long) \_wa1; // fully instantiate type info

void clearCachedFileTimes()

{

long[string] empty;

gCachedFileTimes = empty; // = gCachedFileTimes.init;

}

void removeCachedFileTime(string file)

{

file = canonicalPath(file);

gCachedFileTimes.remove(file);

}

//-----------------------------------------------------------------------------

void getOldestNewestFileTime(string[] files, out long oldest, out long newest, out string oldestFile, out string newestFile)

{

oldest = long.max;

newest = long.min;

foreach(file; files)

{

file = canonicalPath(file);

long ftm;

if(auto ptm = file in gCachedFileTimes)

ftm = \*ptm;

else

{

if(!exists(file))

{

L\_fileNotFound:

oldest = long.min;

newest = long.max;

oldestFile = newestFile = file;

break;

}

version(all)

ftm = timeLastModified(file).stdTime();

else

{

WIN32\_FILE\_ATTRIBUTE\_DATA fad;

if(!GetFileAttributesExW(std.utf.toUTF16z(file), /\*GET\_FILEEX\_INFO\_LEVELS.\*/GetFileExInfoStandard, &fad))

goto L\_fileNotFound;

ftm = \*cast(long\*) &fad.ftLastWriteTime;

}

gCachedFileTimes[file] = ftm;

}

if(ftm > newest)

{

newest = ftm;

newestFile = file;

}

if(ftm < oldest)

{

oldest = ftm;

oldestFile = file;

}

}

}

long getNewestFileTime(string[] files, out string newestFile)

{

string oldestFile;

long oldest, newest;

getOldestNewestFileTime(files, oldest, newest, oldestFile, newestFile);

return newest;

}

long getOldestFileTime(string[] files, out string oldestFile)

{

string newestFile;

long oldest, newest;

getOldestNewestFileTime(files, oldest, newest, oldestFile, newestFile);

return oldest;

}

bool compareCommandFile(string cmdfile, string cmdline)

{

try

{

if(!exists(cmdfile))

return false;

string lastCmd = cast(string)std.file.read(cmdfile);

if (strip(cmdline) != strip(lastCmd))

return false;

}

catch(Exception)

{

return false;

}

return true;

}

bool moveFileToRecycleBin(string fname)

{

SHFILEOPSTRUCT fop;

fop.wFunc = FO\_DELETE;

fop.fFlags = FOF\_NO\_UI | FOF\_NORECURSION | FOF\_FILESONLY | FOF\_ALLOWUNDO;

wstring wname = to!wstring(fname);

wname ~= "\000\000";

fop.pFrom = wname.ptr;

if(SHFileOperation(&fop) != 0)

return false;

return !fop.fAnyOperationsAborted;

}

string shortFilename(string fname)

{

wchar\* sptr;

auto wfname = toUTF16z(fname);

wchar[256] spath;

DWORD len = GetShortPathNameW(wfname, spath.ptr, spath.length);

if(len > spath.length)

{

wchar[] sbuf = new wchar[len];

len = GetShortPathNameW(wfname, sbuf.ptr, cast(DWORD)sbuf.length);

sptr = sbuf.ptr;

}

else

sptr = spath.ptr;

if(len == 0)

return "";

return to!string(sptr[0..len]);

}

string createNewPackageInFolder(string dir, string base)

{

string ndir = normalizeDir(dir);

dir = ndir[0..$-1]; // remove trailing '/'

if (!exists(dir) || !isDir(dir))

return null;

string name = base;

int num = 0;

while(exists(ndir ~ name) || exists(ndir ~ name ~ ".d") || exists(ndir ~ name ~ ".di"))

{

num++;

name = base ~ to!string(num);

}

try

{

mkdir(ndir ~ name);

}

catch(FileException)

{

return null;

}

return name;

}

string[] findDRuntimeFiles(string path, string sub, bool deep, bool cfiles = false, bool internals = false)

{

string[] files;

if(!isExistingDir(path ~ sub))

return files;

foreach(string file; dirEntries(path ~ sub, SpanMode.shallow))

{

if(\_startsWith(file, path))

file = file[path.length .. $];

if (deep && isExistingDir(path ~ file))

{

string[] exclude = [ "[\\internal](file:///\\internal)", "[\\freebsd](file:///\\freebsd)", "[\\linux](file:///\\linux)", "[\\osx](file:///\\osx)", "[\\posix](file:///\\posix)", "[\\solaris](file:///\\solaris)" ];

if (internals)

exclude = exclude[1..$];

if (!any!(e => file.endsWith(e))(exclude))

files ~= findDRuntimeFiles(path, file, deep, cfiles);

continue;

}

string bname = baseName(file);

if(globMatch(bname, "openrj.d"))

continue;

if(globMatch(bname, "minigzip.c") || globMatch(bname, "example.c"))

continue;

if(cfiles)

{

if(globMatch(bname, "\*.c"))

if(!contains(files, file))

files ~= file;

}

else if(globMatch(bname, "\*.d"))

if(string\* pfile = contains(files, file ~ "i"))

\*pfile = file;

else

files ~= file;

else if(globMatch(bname, "\*.di"))

{

// use the d file instead if available

string dfile = "..[\\src\\](file:///\\src\)" ~ file[0..$-1];

if(std.file.exists(path ~ dfile))

file = dfile;

if(!contains(files, file[0..$-1]))

files ~= file;

}

}

return files;

}

///////////////////////////////////////////////////////////////

static struct SymLineInfo

{

string sym;

int firstLine;

uint[] offsets;

}

// map symbol + offset to line in disasm dump

SymLineInfo[string] readDisasmFile(string asmfile)

{

SymLineInfo[string] symInfos;

\_\_gshared static Regex!char resym, resym2, resym3, resym4, reoff, reoff2;

if(resym.ir is null) // dumpbin/llvm-objdump

resym = regex(r"^([A-Za-z\_][^ \t:]\*):$"); // <non numeric symbol>:

if(resym2.ir is null) // obj2asm

resym2 = regex(r"^[ \t]\*assume[ \t]+[Cc][Ss]:([A-Za-z\_][^ \t]\*)[ \t]\*$"); // assume CS:<non numeric symbol>

if(resym3.ir is null) // objconv

resym3 = regex(r"^([A-Za-z\_][^ \t]\*)[ \t]+PROC[ \t]+NEAR[ \t]\*$"); // <non numeric symbol> PROC NEAR

if(resym4.ir is null) // gcc-objdump

resym4 = regex(r"^[0-9A-Fa-f]+[ \t]\*\<([A-Za-z\_][^>]\*)\>:[ \t]\*$"); // 000000 <non numeric symbol>

if(reoff.ir is null)

reoff = regex(r"^([0-9A-Fa-f]+):.\*$"); // <hex number>:

if(reoff2.ir is null)

reoff2 = regex(r"[^;]\*;[ \t:]\*([0-9A-Fa-f]+) \_.\*$"); // ; <hex number> \_

int ln = 0;

SymLineInfo info;

File asmf = File(asmfile);

foreach(line; asmf.byLine())

{

ln++;

if (line.length == 0)

{

// intermediate lines in objconv output happen to contain a \t

if (info.offsets.length)

{

symInfos[info.sym] = info;

info.sym = null;

info.offsets = null;

}

continue;

}

line = toUTF8Safe(line);

line = strip(line);

auto rematch = match(line, resym);

if (rematch.empty())

rematch = match(line, resym2);

if (rematch.empty())

rematch = match(line, resym3);

if (rematch.empty())

rematch = match(line, resym4);

if (!rematch.empty())

{

if (info.offsets.length)

symInfos[info.sym] = info;

info.sym = rematch.captures[1].idup;

info.firstLine = ln;

info.offsets = null;

continue;

}

rematch = match(line, reoff);

if (rematch.empty())

rematch = match(line, reoff2);

if (!rematch.empty())

{

uint off = rematch.captures[1].to!uint(16);

info.offsets ~= off;

}

else if (info.sym.length)

{

if (info.offsets.length)

info.offsets ~= info.offsets[$-1];

else

info.offsets ~= 0;

}

}

if (info.offsets.length)

symInfos[info.sym] = info;

return symInfos;

}

unittest

{

string dumpbin = r"

Dump of file Debug\winmain.obj

File Type: COFF OBJECT

WinMain:

0000000000000000: 55 push rbp

0000000000000001: 48 8B EC mov rbp,rsp

00

0000000000000004: 48 83 EC 28 sub rsp,28h

; obj2asm style

assume CS:\_D7winmain9myWinMainFPvPvPaiZi

0000000000000000: 55 push rbp

0000000000000001: 48 8B EC mov rbp,rsp

0000000000000004: 48 83 EC 30 sub rsp,30h

; objconv style

\_WinMain@16 PROC NEAR

; COMDEF \_WinMain@16

push ebp ; 0000 \_ 55

mov ebp, esp ; 0001 \_ 8B. EC

ASSUME fs:NOTHING

push 48 ; 0003 \_ 6A, 30

" /\* explicite trailing spaces before nl \*/ "

; Note: No jump seems to point here

mov ecx, offset FLAT:?\_009 ; 0005 \_ B9, 00000000(segrel)

Disassembly of section .text: GNU objdump

0000000000000000 <\_foo>:

0:        55         push %rbp

1:        48 89 e5         mov %rsp,%rbp

";

auto deleteme = "deleteme";

std.file.write(deleteme, dumpbin);

scope(exit) std.file.remove(deleteme);

auto symInfo = readDisasmFile(deleteme);

assert(symInfo.length == 4);

assert(symInfo["WinMain"].firstLine == 6);

assert(symInfo["WinMain"].offsets.length == 4);

assert(symInfo["\_D7winmain9myWinMainFPvPvPaiZi"].offsets.length == 3);

assert(symInfo["\_WinMain@16"].firstLine == 19);

assert(symInfo["\_WinMain@16"].offsets.length == 8);

assert(symInfo["\_WinMain@16"].offsets[3] == 1);

assert(symInfo["\_foo"].offsets.length == 2);

}

struct LineInfo

{

string sym;

int offset;

}

// map line in source to symbol and offset in object file

LineInfo[] readLineInfoFile(string linefile, string srcfile)

{

\_\_gshared static Regex!char reoffline;

if(reoffline.ir is null)

reoffline = regex(r"^Off 0x([0-9A-Fa-f]+): \*Line ([0-9]+)$"); // Off 0x%x: Line %d

srcfile = toLower(normalizePath(srcfile));

string sym;

bool curfile;

LineInfo[] lineInfos;

File linef = File(linefile);

foreach(line; linef.byLine())

{

line = toUTF8Safe(line);

line = strip(line);

if (line.startsWith("Sym:"))

sym = strip(line[4 .. $]).idup;

else if (line.startsWith("File:"))

{

auto file = toLower(normalizePath(strip(line[5 .. $])));

if (srcfile.contains('\\') != file.contains('\\'))

{

srcfile = srcfile[lastIndexOf(srcfile, '[\\')+1](file:///\\')+1) .. $];

file = file[lastIndexOf(file, '[\\')+1](file:///\\')+1) .. $];

}

curfile = (srcfile == file);

}

else if (curfile)

{

auto rematch = match(line, reoffline);

if (!rematch.empty())

{

int off = rematch.captures[1].to!uint(16);

int ln = rematch.captures[2].to!uint(10);

if (ln >= lineInfos.length)

lineInfos.length = ln + 100;

if (lineInfos[ln].sym.ptr is null)

lineInfos[ln] = LineInfo(sym, off);

}

}

}

return lineInfos;

}

unittest

{

string dumpline = r"

Sym: WinMain

File: WindowsApp1\winmain.d

Off 0x0: Line 7

Off 0x23: Line 9

Off 0x2a: Line 18

Off 0x37: Line 20

Sym: \_D7winmain7WinMainWPvPvPaiZ2ehMFC6object9ThrowableZv

File: WindowsApp1\winmain.d

Off 0x0: Line 11

Off 0xc: Line 13

Off 0x19: Line 14

Off 0xfffffffe: Line 16" /\* bad offset generated by DMD \*/ "

";

auto deleteme = "deleteme";

std.file.write(deleteme, dumpline);

scope(exit) std.file.remove(deleteme);

auto infos = readLineInfoFile(deleteme, r"WindowsApp1\winmain.d");

assert(infos.length > 20);

assert(infos[7].sym == "WinMain" && infos[7].offset == 0);

assert(infos[20].sym == "WinMain" && infos[20].offset == 0x37);

assert(infos[13].sym == "\_D7winmain7WinMainWPvPvPaiZ2ehMFC6object9ThrowableZv" && infos[13].offset == 0xc);

assert(infos[14].sym == "\_D7winmain7WinMainWPvPvPaiZ2ehMFC6object9ThrowableZv" && infos[14].offset == 0x19);

}

// This file is part of Visual D

//

// Visual D integrates the D programming language into Visual Studio

// Copyright (c) 2010-2012 by Rainer Schuetze, All Rights Reserved

//

// Distributed under the Boost Software License, Version 1.0.

// See accompanying file LICENSE\_1\_0.txt or copy at <http://www.boost.org/LICENSE_1_0.txt>

module visuald.getmsobj;

import visuald.register;

import visuald.hierutil;

import visuald.fileutil;

import visuald.windows;

import stdext.httpget;

import stdext.path;

import sdk.win32.winreg;

import std.path;

import std.conv;

import std.file;

import core.stdc.stdlib;

// for msobj80.dll

// <http://download.microsoft.com/download/2/E/9/2E911956-F90F-4BFB-8231-E292A7B6F287/GRMSDK_EN_DVD.iso>

// FL\_msobj71\_dll\_1\_60033\_x86\_ln.3643236F\_FC70\_11D3\_A536\_0090278A1BB8

// in vc\_stdx86.cab

// for msobj100.dll

//

// <http://download.microsoft.com/download/1/E/5/1E5F1C0A-0D5B-426A-A603-1798B951DDAE/VS2010Express1.iso>

// FL\_msobj71\_dll\_1\_60033\_x86\_ln.3643236F\_FC70\_11D3\_A536\_0090278A1BB8

// in vs\_setup.cab

// in lxpvc.exe (msi)

// or

// <http://download.microsoft.com/download/4/0/E/40EFE5F6-C7A5-48F7-8402-F3497FABF888/X16-42555VS2010ProTrial1.iso>

// FL\_msobj71\_dll\_1\_60033\_x86\_ln.3643236F\_FC70\_11D3\_A536\_0090278A1BB8

// in cab14.cab

HRESULT VerifyMSObjectParser(wstring winstallDir)

{

debug UtilMessageBox("VerifyMSObj(dir=" ~ to!string(winstallDir) ~ ")", MB\_OK, "Visual D Installer");

if(!winstallDir.length)

return S\_FALSE;

string installDir = to!string(winstallDir);

HRESULT checkMSObj(string ver, string url, ulong cab\_start, ulong cab\_length)

{

debug UtilMessageBox("checkMSObj(ver=" ~ ver ~ ")", MB\_OK, "Visual D Installer");

string mspdb = "mspdb" ~ ver ~ ".dll";

string absmspdb = buildPath(installDir, mspdb);

if(exists(absmspdb))

{

debug UtilMessageBox(absmspdb ~ " exists", MB\_OK, "Visual D Installer");

string msobj = "msobj" ~ ver ~ ".dll";

string absmsobj = buildPath(installDir, msobj);

if(exists(absmsobj))

return S\_OK;

int res = UtilMessageBox("The file " ~ msobj ~ "\n"

"is missing in your Visual Studio installation.\n"

"Would you like to download it from the Windows 7 SDK?",

MB\_YESNO, "Visual Studio Shell detected");

if(res == IDYES)

{

string tmp\_cab = buildPath(tempDir(), "vd\_install\_from\_w7sdk.cab");

for (;;)

{

try

{

auto length = httpget("download.microsoft.com", 80, url, tmp\_cab, cab\_start, cab\_length);

if(length != cab\_length)

throw new Exception("Unexpected file length");

}

catch(Exception e)

{

res = UtilMessageBox("Error while downloading:\n" ~ e.msg ~ "\n",

MB\_ABORTRETRYIGNORE, "Visual D Installer");

if(res == IDABORT)

return E\_ABORT;

if(res == IDIGNORE)

return S\_OK;

}

break;

}

string srcfile = "FL\_msobj71\_dll\_1\_60033\_x86\_ln.3643236F\_FC70\_11D3\_A536\_0090278A1BB8";

string cmd = "expand " ~ shortFilename(tmp\_cab) ~ " -f:" ~ srcfile ~ " " ~ shortFilename(installDir);

for(;;)

{

string logfile = tmp\_cab ~ ".expand\_log";

//scope(exit) if (exists(logfile)) remove(logfile);

std.file.write(logfile, cmd);

if(system((cmd ~ " >> " ~ logfile ~ " 2>&1").ptr) != 0)

{

string output = readText(logfile);

res = UtilMessageBox("Error while expanding:\n" ~ cmd ~ "\n" ~ output,

MB\_ABORTRETRYIGNORE, "Visual D Installer");

if(res == IDABORT)

return E\_ABORT;

if(res == IDIGNORE)

return S\_OK;

}

break;

}

try

{

rename(buildPath(installDir, srcfile), absmsobj);

}

catch(Exception e)

{

UtilMessageBox("Error while renaming:\n" ~ e.msg, MB\_OK, "Visual D Installer");

return S\_FALSE;

}

}

}

return S\_OK;

}

HRESULT hr;

hr = checkMSObj("80", "/download/2/E/9/2E911956-F90F-4BFB-8231-E292A7B6F287/GRMSDK\_EN\_DVD.iso",

0x59b07000, 0x29524dc);

if(hr == S\_OK)

hr = checkMSObj("100", "/download/4/0/E/40EFE5F6-C7A5-48F7-8402-F3497FABF888/X16-42555VS2010ProTrial1.iso",

0x1b03000, 14\_039\_060);

return hr;

}

// This file is part of Visual D

//

// Visual D integrates the D programming language into Visual Studio

// Copyright (c) 2010 by Rainer Schuetze, All Rights Reserved

//

// Distributed under the Boost Software License, Version 1.0.

// See accompanying file LICENSE\_1\_0.txt or copy at <http://www.boost.org/LICENSE_1_0.txt>

module visuald.help;

import visuald.fileutil;

import visuald.dpackage;

import visuald.pkgutil;

import visuald.hierutil;

import visuald.comutil;

import dte2 = sdk.vsi.dte80;

import stdext.path;

import std.file;

import std.path;

import std.string;

import std.utf;

import std.uri;

import std.conv;

import std.array;

//////////////////////////////////////////////////////////////////////

static string[][string] tags;

alias AssociativeArray!(string, string[]) \_wa1; // fully instantiate type info

static bool[string] searchAnchors(string file)

{

bool[string] names;

string s = to!string(std.file.read(file));

fulltext:

for(size\_t pos = 0; pos < s.length; )

{

dchar ch = decode(s, pos);

if(ch == '<')

{

if(s[pos..$].startsWith("a name=\""))

{

auto p = s[pos+8..$].indexOf('\"');

if(p < 0)

break fulltext;

string name = s[pos+8 .. pos+8 + p];

names[name] = true;

pos += 8 + p + 1;

}

while(ch != '>' && pos < s.length)

{

ch = decode(s, pos);

if(ch == '\"')

{

auto p = s[pos..$].indexOf('\"');

if(p < 0)

break fulltext;

pos += p + 1;

}

}

}

else if(ch == '\"')

{

auto p = s[pos..$].indexOf('\"');

if(p < 0)

break fulltext;

pos += p + 1;

}

}

return names;

}

void loadTags()

{

string installdir = normalizeDir(Package.GetGlobalOptions().DMD.InstallDir) ~ "html/d/";

if(!std.file.exists(installdir ~ "index.html"))

{

writeToBuildOutputPane("no documentation found at " ~ installdir);

return;

}

tags = tags.init;

foreach(string file; dirEntries(installdir, SpanMode.depth))

{

try

{

string bname = baseName(file);

if(globMatch(bname, "\*.html"))

{

auto names = searchAnchors(file);

foreach(name, b; names)

tags[name] ~= file;

}

}

catch(Exception e)

{

// bad file access, utf8 exception, etc

writeToBuildOutputPane("failed to read " ~ file);

}

}

}

string replacePath(string s, string href, string path)

{

string url = "[file://](NULL)" ~ replace(path, "\\", "/");

string t;

for( ; ; )

{

int pos = s.indexOf(href);

if(pos < 0)

break;

t ~= s[0..pos + href.length];

s = s[pos + href.length .. $];

if(!s.startsWith("http:/"))

t ~= url;

}

t ~= s;

return t;

}

string replaceRef(string s, string path)

{

s = replacePath(s, `href="`, path);

s = replacePath(s, `src="`, path);

return s;

}

string createDisambiguationPage(string word, string[] files)

{

string installdir = normalizeDir(Package.GetGlobalOptions().DMD.InstallDir) ~ "html/d/";

string fallback = `<html lang="en-US"><head></head><body class="hyphenate"><div id="content"></div>`

`<div id="footernav"></div></body></html>`;

string html = fallback;

string idxfile = installdir ~ "index.html";

if(std.file.exists(idxfile))

html = to!string(std.file.read(idxfile));

string start = `<div id="content">`;

string footer = `<div id="footernav">`;

int ps = html.indexOf(start);

int pe = html.indexOf(footer);

if(ps < 0 || pe < ps)

{

html = fallback;

ps = html.indexOf(start);

pe = html.indexOf(footer);

}

string gen = "<p>There are multiple pages commenting on &quot;" ~ word ~ "&quot;</p><ul>\n";

foreach(f; files)

{

string url = std.uri.encode("file://" ~ replace(f, "\\", "/") ~ "#" ~ word);

string name = replace(stripExtension(baseName(f)), "\_", ".");

gen ~= `<li><a href="` ~ url ~ `">` ~ name ~ "</a></li>\n";

}

gen ~= "</ul>";

string beg = replaceRef(html[0..ps + start.length], installdir);

string end = replaceRef(html[pe..$], installdir);

string nhtml = beg ~ gen ~ "</div>" ~ end;

wchar[MAX\_PATH] path;

uint len = GetTempPath(MAX\_PATH, path.ptr);

string fname = normalizeDir(to\_string(path.ptr, len)) ~ "vd\_disambiguation.html";

std.file.write(fname, nhtml);

return fname;

}

bool openHelp(string word)

{

static bool triedLoad;

if(!triedLoad) // (tags.length == 0) no longer works ;-((

{

triedLoad = true;

loadTags();

}

string url;

auto files = word in tags;

void tryAlternative(string alt)

{

if(!files)

{

files = alt in tags;

if(files)

word = alt;

}

}

tryAlternative(capitalize(word) ~ "Statement");

tryAlternative(capitalize(word) ~ "Declaration");

tryAlternative(capitalize(word) ~ "Expression");

if(word == "unittest") tryAlternative("UnitTest");

if(word == "function" || word == "delegate") tryAlternative("closures");

if(word == "\_\_traits" || word == "traits") tryAlternative("TraitsExpression");

if(files)

{

string file;

if((\*files).length == 1)

file = (\*files)[0] ~ "#" ~ word;

else

file = createDisambiguationPage(word, \*files);

url = std.uri.encode("file://" ~ replace(file, "\\", "/"));

}

if(url.length == 0)

return false;

if(dte2.DTE2 spvsDTE = GetDTE())

{

scope(exit) release(spvsDTE);

spvsDTE.ExecuteCommand("View.WebBrowser"w.ptr, \_toUTF16z(url));

}

return true;

}

// This file is part of Visual D

//

// Visual D integrates the D programming language into Visual Studio

// Copyright (c) 2010 by Rainer Schuetze, All Rights Reserved

//

// Distributed under the Boost Software License, Version 1.0.

// See accompanying file LICENSE\_1\_0.txt or copy at <http://www.boost.org/LICENSE_1_0.txt>

module visuald.hierarchy;

import visuald.windows;

import sdk.win32.commctrl;

import std.string;

import std.path;

import std.file;

import std.utf;

import std.array;

import std.algorithm;

import std.process : browse;

import stdext.path;

import stdext.file;

import sdk.port.vsi;

import sdk.vsi.vsshell;

import sdk.vsi.vsshell80;

import sdk.vsi.fpstfmt;

import sdk.vsi.ivssccmanager2;

//import vsshlids;

import visuald.comutil;

import visuald.logutil;

import visuald.lexutil;

import visuald.trackprojectdocument;

import visuald.hierutil;

import visuald.chiernode;

import visuald.chiercontainer;

import visuald.propertypage;

import visuald.fileutil;

import visuald.stringutil;

import visuald.dimagelist;

import visuald.config;

import visuald.pkgutil;

import visuald.dproject;

import visuald.dpackage;

import visuald.dllmain;

///////////////////////////////////////////////////////////////////////////////

class CFileNode : CHierNode,

ISpecifyPropertyPages,

IVsGetCfgProvider

{

static const GUID iid = { 0x3fc35781, 0xfbb0, 0x42b6, [ 0xa2, 0x9b, 0x42, 0xdf, 0xa4, 0x96, 0x39, 0x2 ] };

this(string filename)

{

mFilename = filename;

SetName(baseName(filename));

}

override HRESULT QueryInterface(in IID\* riid, void\*\* pvObject)

{

if(queryInterface!(CFileNode) (this, riid, pvObject))

return S\_OK;

if(queryInterface!(ISpecifyPropertyPages) (this, riid, pvObject))

return S\_OK;

if(queryInterface!(IVsGetCfgProvider) (this, riid, pvObject))

return S\_OK;

return super.QueryInterface(riid, pvObject);

}

// ISpecifyPropertyPages

override int GetPages( /\* [out] \*/ CAUUID \*pPages)

{

mixin(LogCallMix);

return PropertyPageFactory.GetCommonPages(pPages);

}

// IVsGetCfgProvider

override int GetCfgProvider(IVsCfgProvider\* pCfgProvider)

{

if(Project prj = cast(Project) GetCVsHierarchy())

return prj.GetCfgProvider(pCfgProvider);

return E\_NOINTERFACE;

}

// Property functions

override int GetProperty(VSHPROPID propid, out VARIANT var)

{

switch(propid)

{

case VSHPROPID\_Name:

case VSHPROPID\_SaveName:

var.vt = VT\_BSTR;

var.bstrVal = allocBSTR(GetName());

return S\_OK;

case VSHPROPID\_StateIconIndex:

var.vt = VT\_I4;

var.lVal = STATEICON\_NOSTATEICON;

if(IVsSccManager2 sccmgr = queryService!(SVsSccManager, IVsSccManager2)())

{

scope(exit) release(sccmgr);

auto path = \_toUTF16z(GetFullPath());

VsStateIcon icon;

DWORD sccStatus;

if(sccmgr.GetSccGlyph(1, &path, &icon, &sccStatus) == S\_OK)

var.lVal = icon;

}

return S\_OK;

default:

return super.GetProperty(propid, var);

}

}

override int SetProperty(VSHPROPID propid, in VARIANT var)

{

switch(propid)

{

case VSHPROPID\_EditLabel:

if(var.vt != VT\_BSTR)

return returnError(E\_INVALIDARG);

string newname = to\_string(var.bstrVal);

return Rename(newname);

default:

return super.SetProperty(propid, var);

}

}

override HRESULT GetGuidProperty(VSHPROPID propid, out GUID pGuid)

{

switch (propid)

{

case VSHPROPID\_TypeGuid:

// we represent physical file on disk so

// return the corresponding guid defined in vsshell.idl

pGuid = GUID\_ItemType\_PhysicalFile;

break;

default:

return DISP\_E\_MEMBERNOTFOUND;

}

return S\_OK;

}

HRESULT Rename(string newname)

{

string oldpath = GetFullPath();

string newpath = normalizeDir(dirName(oldpath)) ~ newname;

if(toLower(newname) == toLower(mFilename))

return S\_OK;

bool wasOpen;

int line = -1;

int col = 0;

GetDocInfo(&wasOpen, null, null, null);

if (wasOpen)

if (auto tv = Package.GetLanguageService().GetView(oldpath))

tv.GetCaretPos(&line, &col);

if(HRESULT hr = CloseDoc(SLNSAVEOPT\_PromptSave))

return hr;

tryWithExceptionToBuildOutputPane(()

{

std.file.rename(oldpath, newpath);

string projDir = GetCVsHierarchy().GetProjectDir();

mFilename = makeRelative(newpath, projDir);

SetName(baseName(mFilename));

GetCVsHierarchy().GetProjectNode().SetProjectFileDirty(true);

if (wasOpen)

if(CVsHierarchy hier = GetCVsHierarchy())

{

hier.OpenDoc(this, false, false, true);

if (auto tv = Package.GetLanguageService().GetView(newpath))

if (line >= 0)

tv.SetCaretPos(line, col);

}

});

return S\_OK;

}

override string GetFullPath()

{

if(isAbsolute(mFilename))

return mFilename;

string root = GetRootNode().GetFullPath();

root = dirName(root);

return removeDotDotPath(root ~ "\\" ~ mFilename);

}

string GetFilename()

{

return mFilename;

}

bool GetPerConfigOptions()

{

return mPerConfigOptions;

}

void SetPerConfigOptions(bool perConfig)

{

mPerConfigOptions = perConfig;

if(!mPerConfigOptions)

mConfigOptions = mConfigOptions.init;

if(CVsHierarchy hier = GetCVsHierarchy())

hier.OnPropertyChanged(this, VSHPROPID\_IconIndex, 0);

}

string GetTool(string cfg)

{

return getOptions(cfg).mTool;

}

void SetTool(string cfg, string tool)

{

createOptions(cfg).mTool = tool;

if(CVsHierarchy hier = GetCVsHierarchy())

hier.OnPropertyChanged(this, VSHPROPID\_IconIndex, 0);

}

string GetDependencies(string cfg)

{

return getOptions(cfg).mDependencies;

}

void SetDependencies(string cfg, string dep)

{

createOptions(cfg).mDependencies = dep;

}

string GetOutFile(string cfg)

{

return getOptions(cfg).mOutFile;

}

void SetOutFile(string cfg, string file)

{

createOptions(cfg).mOutFile = file;

}

string GetCustomCmd(string cfg)

{

return getOptions(cfg).mCustomCmd;

}

void SetCustomCmd(string cfg, string cmd)

{

createOptions(cfg).mCustomCmd = cmd;

}

string GetAdditionalOptions(string cfg)

{

return getOptions(cfg).mAddOpt;

}

void SetAdditionalOptions(string cfg, string opt)

{

createOptions(cfg).mAddOpt = opt;

}

bool GetLinkOutput(string cfg)

{

return getOptions(cfg).mLinkOut;

}

void SetLinkOutput(string cfg, bool lnk)

{

createOptions(cfg).mLinkOut = lnk;

}

bool GetUptodateWithSameTime(string cfg)

{

return getOptions(cfg).mUptodateWithSameTime;

}

void SetUptodateWithSameTime(string cfg, bool uptodateWithSameTime)

{

createOptions(cfg).mUptodateWithSameTime = uptodateWithSameTime;

}

Options[string] GetConfigOptions() { return mConfigOptions; }

override int DoDefaultAction()

{

if(CVsHierarchy hier = GetCVsHierarchy())

return hier.OpenDoc(this, false, false, true);

return S\_OK;

}

override uint GetContextMenu() { return IDM\_VS\_CTXT\_ITEMNODE; }

override int QueryStatus(

/\* [unique][in] \*/ in GUID \*pguidCmdGroup,

/\* [in] \*/ ULONG cCmds,

/\* [out][in][size\_is] \*/ OLECMD\* prgCmds,

/\* [unique][out][in] \*/ OLECMDTEXT \*pCmdText)

{

OLECMD\* Cmd = prgCmds;

HRESULT hr = S\_OK;

bool fSupported = false;

bool fEnabled = false;

bool fInvisible = false;

if(\*pguidCmdGroup == CMDSETID\_StandardCommandSet97)

{

switch(Cmd.cmdID)

{

case cmdidOpenWith:

case cmdidOpen:

fSupported = true;

fEnabled = true;

break;

case cmdidViewCode:

fSupported = true;

fEnabled = Config.IsResource(this);

break;

default:

hr = OLECMDERR\_E\_NOTSUPPORTED;

break;

}

}

else

{

hr = OLECMDERR\_E\_NOTSUPPORTED;

}

if (SUCCEEDED(hr) && fSupported)

{

Cmd.cmdf = OLECMDF\_SUPPORTED;

if (fInvisible)

Cmd.cmdf |= OLECMDF\_INVISIBLE;

else if (fEnabled)

Cmd.cmdf |= OLECMDF\_ENABLED;

}

if (hr == OLECMDERR\_E\_NOTSUPPORTED)

hr = super.QueryStatus(pguidCmdGroup, cCmds, prgCmds, pCmdText);

return hr;

}

override int Exec(

/\* [unique][in] \*/ in GUID \*pguidCmdGroup,

/\* [in] \*/ DWORD nCmdID,

/\* [in] \*/ DWORD nCmdexecopt,

/\* [unique][in] \*/ in VARIANT \*pvaIn,

/\* [unique][out][in] \*/ VARIANT \*pvaOut)

{

int hr = OLECMDERR\_E\_NOTSUPPORTED;

if(\*pguidCmdGroup == CMDSETID\_StandardCommandSet97)

{

switch(nCmdID)

{

case cmdidOpenWith:

hr = GetCVsHierarchy().OpenDoc(this, false, true, true);

break;

case cmdidOpen:

hr = GetCVsHierarchy().OpenDoc(this, false, false, true);

break;

case cmdidViewCode:

hr = GetCVsHierarchy().OpenDoc(this, false, false, true, &LOGVIEWID\_Code);

break;

default:

break;

}

}

if (hr == OLECMDERR\_E\_NOTSUPPORTED)

hr = super.Exec(pguidCmdGroup, nCmdID, nCmdexecopt, pvaIn, pvaOut);

return hr;

}

HRESULT GetRDTDocumentInfo(

/\* [in] \*/ string pszDocumentName,

/\* [out] \*/ IVsHierarchy\* ppIVsHierarchy /\* = NULL \*/,

/\* [out] \*/ VSITEMID\* pitemid /\* = NULL \*/,

/\* [out] \*/ IVsPersistDocData\* ppIVsPersistDocData /\* = NULL \*/,

/\* [out] \*/ VSDOCCOOKIE\* pVsDocCookie /\* = NULL \*/)

{

// Get the document info.

IVsRunningDocumentTable pRDT = queryService!(IVsRunningDocumentTable);

if(!pRDT)

return E\_FAIL;

scope(exit) release(pRDT);

auto docname = \_toUTF16z(pszDocumentName);

IVsHierarchy srpIVsHierarchy;

VSITEMID vsItemId = VSITEMID\_NIL;

IUnknown srpIUnknown;

VSDOCCOOKIE vsDocCookie = VSDOCCOOKIE\_NIL;

HRESULT hr = pRDT.FindAndLockDocument(

/\* [in] VSRDTFLAGS dwRDTLockType \*/ RDT\_NoLock,

/\* [in] LPCOLESTR pszMkDocument \*/ docname,

/\* [out] IVsHierarchy \*\*ppHier \*/ &srpIVsHierarchy,

/\* [out] VSITEMID \*pitemid \*/ &vsItemId,

/\* [out] IUnknown \*\*ppunkDocData \*/ &srpIUnknown,

/\* [out] VSCOOKIE \*pdwCookie \*/ &vsDocCookie);

// FindAndLockDocument returns S\_FALSE if the doc is not in the RDT

if (FAILED(hr))

return hr;

scope(exit) release(srpIUnknown);

scope(exit) release(srpIVsHierarchy);

// now return the requested info

if (ppIVsHierarchy && srpIVsHierarchy)

\*ppIVsHierarchy = addref(srpIVsHierarchy);

if (pitemid)

\*pitemid = vsItemId;

if (ppIVsPersistDocData && srpIUnknown)

srpIUnknown.QueryInterface(&IVsPersistDocData.iid, cast(void\*\*)ppIVsPersistDocData);

if (pVsDocCookie)

\*pVsDocCookie = vsDocCookie;

return S\_OK;

}

HRESULT GetDocInfo(

/\* [out, opt] \*/ bool\* pfOpen, // true if the doc is opened

/\* [out, opt] \*/ bool\* pfDirty, // true if the doc is dirty

/\* [out, opt] \*/ bool\* pfOpenByUs, // true if opened by our project

/\* [out, opt] \*/ VSDOCCOOKIE\* pVsDocCookie)// VSDOCCOOKIE if open

{

if (!pfOpen && !pfDirty && !pfOpenByUs && !pVsDocCookie)

return S\_OK;

if (pfOpen) \*pfOpen = false;

if (pfDirty) \*pfDirty = false;

if (pfOpenByUs) \*pfOpenByUs = false;

if (pVsDocCookie) \*pVsDocCookie = VSDOCCOOKIE\_NIL;

HRESULT hr = S\_OK;

string strFullName = GetFullPath();

IVsHierarchy srpIVsHierarchy;

IVsPersistDocData srpIVsPersistDocData;

VSITEMID vsitemid = VSITEMID\_NIL;

VSDOCCOOKIE vsDocCookie = VSDOCCOOKIE\_NIL;

hr = GetRDTDocumentInfo(

/\* [in] LPCTSTR pszDocumentName \*/ strFullName,

/\* [out] IVsHierarchy\*\* ppIVsHierarchy \*/ &srpIVsHierarchy,

/\* [out] VSITEMID\* pitemid \*/ &vsitemid,

/\* [out] IVsPersistDocData\*\* ppIVsPersistDocData\*/ &srpIVsPersistDocData,

/\* [out] VSDOCCOOKIE\* pVsDocCookie \*/ &vsDocCookie);

if (FAILED(hr))

return hr;

scope(exit) release(srpIVsHierarchy);

scope(exit) release(srpIVsPersistDocData);

if (!srpIVsHierarchy || (vsDocCookie == VSDOCCOOKIE\_NIL))

return S\_OK;

if (pfOpen)

\*pfOpen = TRUE;

if (pVsDocCookie)

\*pVsDocCookie = vsDocCookie;

if (pfOpenByUs)

{

// check if the doc is opened by another project

IVsHierarchy pMyHier = GetCVsHierarchy().GetIVsHierarchy();

IUnknown punkMyHier;

pMyHier.QueryInterface(&IID\_IUnknown, cast(void \*\*)&punkMyHier);

IUnknown punkRDTHier;

srpIVsHierarchy.QueryInterface(&IID\_IUnknown, cast(void \*\*)&punkRDTHier);

if (punkRDTHier is punkMyHier)

\*pfOpenByUs = true;

release(punkMyHier);

release(punkRDTHier);

}

if (pfDirty && srpIVsPersistDocData)

{

BOOL dirty;

hr = srpIVsPersistDocData.IsDocDataDirty(&dirty);

\*pfDirty = dirty != 0;

}

return S\_OK;

}

HRESULT SaveDoc(/\* [in] \*/ VSSLNSAVEOPTIONS grfSaveOpts)

{

HRESULT hr = S\_OK;

bool fOpen = FALSE;

bool fDirty = TRUE;

bool fOpenByUs = FALSE;

VSDOCCOOKIE vsDocCookie = VSDOCCOOKIE\_NIL;

hr = GetDocInfo(

/\* [out, opt] BOOL\* pfOpen \*/ &fOpen, // true if the doc is opened

/\* [out, opt] BOOL\* pfDirty \*/ &fDirty, // true if the doc is dirty

/\* [out, opt] BOOL\* pfOpenByUs \*/ &fOpenByUs, // true if opened by our project

/\* [out, opt] VSDOCCOOKIE\* pVsDocCookie\*/ &vsDocCookie);// VSDOCCOOKIE if open

if (FAILED(hr) || /\*!fOpenByUs ||\*/ vsDocCookie == VSDOCCOOKIE\_NIL)

return hr;

IVsSolution pIVsSolution = queryService!(IVsSolution);

if(!pIVsSolution)

return E\_FAIL;

scope(exit) pIVsSolution.Release();

return pIVsSolution.SaveSolutionElement(

/\* [in] VSSLNSAVEOPTIONS grfSaveOpts\*/ grfSaveOpts,

/\* [in] IVsHierarchy \*pHier \*/ null,

/\* [in] VSCOOKIE docCookie \*/ vsDocCookie);

}

HRESULT CloseDoc(/\* [in] \*/ VSSLNCLOSEOPTIONS grfCloseOpts)

{

HRESULT hr = S\_OK;

bool fOpen = false;

bool fOpenByUs = false;

VSDOCCOOKIE vsDocCookie = VSDOCCOOKIE\_NIL;

hr = GetDocInfo(

/\* [out, opt] BOOL\* pfOpen \*/ &fOpen, // true if the doc is opened

/\* [out, opt] BOOL\* pfDirty \*/ null, // true if the doc is dirty

/\* [out, opt] BOOL\* pfOpenByUs \*/ &fOpenByUs, // true if opened by our project

/\* [out, opt] VSDOCCOOKIE\* pVsDocCookie\*/ &vsDocCookie);// VSDOCCOOKIE if open

if (FAILED(hr) || !fOpenByUs || vsDocCookie == VSDOCCOOKIE\_NIL)

return hr;

IVsSolution pIVsSolution = queryService!(IVsSolution);

if(!pIVsSolution)

return E\_FAIL;

scope(exit) pIVsSolution.Release();

// may return E\_ABORT if prompt is cancelled

return pIVsSolution.CloseSolutionElement(

/\* [in] VSSLNCLOSEOPTIONS grfCloseOpts \*/ grfCloseOpts,

/\* [in] IVsHierarchy \*pHier \*/ null,

/\* [in] VSCOOKIE docCookie \*/ vsDocCookie);

}

CFileNode cloneDeep()

{

CFileNode n = clone(this);

n.mConfigOptions = mConfigOptions.dup;

return n;

}

private:

Options\* \_getOptions(string cfg, bool create)

{

if(mPerConfigOptions && cfg.length)

{

if(Options\* opt = cfg in mConfigOptions)

return opt;

else if(create)

{

mConfigOptions[cfg] = mGlobalOptions;

return cfg in mConfigOptions;

}

}

return &mGlobalOptions;

}

Options\* getOptions(string cfg)

{

return \_getOptions(cfg, false);

}

Options\* createOptions(string cfg)

{

return \_getOptions(cfg, true);

}

static struct Options

{

string mTool;

string mDependencies;

string mOutFile;

string mCustomCmd;

string mAddOpt;

bool mLinkOut;

bool mUptodateWithSameTime;

}

Options mGlobalOptions;

Options[string] mConfigOptions;

string mFilename; // relative or absolute

bool mPerConfigOptions;

}

// virtual folder

class CFolderNode : CHierContainer

{

this(string name = "")

{

SetName(name);

SetIsSortedList(hierContainerIsSorted);

}

// VSHPROPID\_EditLabel

override int GetEditLabel(BSTR \*ppEditLabel)

{

\*ppEditLabel = allocBSTR(GetName());

return S\_OK;

}

override int SetEditLabel(in BSTR pEditLabel)

{

string label = to\_string(pEditLabel);

// only rename folder for package if no files in project folder

if(searchNode(this, (CHierNode n) { return cast(CFileNode) n !is null; }) is null)

{

string dir = GuessFolderPath();

if (std.file.exists(dir) && std.file.isDir(dir))

{

string newdir = normalizeDir(dirName(dir)) ~ label;

scope dg = (){

std.file.rename(dir, newdir);

};

if (!tryWithExceptionToBuildOutputPane(dg))

return S\_FALSE;

}

}

SetName(label);

GetCVsHierarchy().OnPropertyChanged(this, VSHPROPID\_Name, 0);

return S\_OK;

}

string GuessPackageName()

{

string pkgname = \_GuessPackageName(true, null);

if(pkgname.endsWith("."))

pkgname = pkgname[0..$-1];

if(pkgname.startsWith("."))

pkgname = pkgname[1..$];

return pkgname;

}

// package always comes with trailing '.'

string \_GuessPackageName(bool recurseUp, CFolderNode exclude)

{

static string stripModule(string mod)

{

auto pos = lastIndexOf(mod, '.');

if(pos >= 0)

return mod[0..pos+1];

return ".";

}

static string stripPackage(string pkg, string folder)

{

assert(pkg.length && pkg[$-1] == '.');

auto pos = lastIndexOf(pkg[0..$-1], '.');

if(pos >= 0 && icmp(pkg[pos+1 .. $-1], folder) == 0)

return pkg[0..pos+1];

if(pos >= 0)

return pkg;

return ".";

}

// check files in folder

for(CHierNode pNode = GetHead(); pNode; pNode = pNode.GetNext())

if(auto file = cast(CFileNode) pNode)

{

string tool = file.GetTool(null);

if(tool == "DMD" || (tool == "" && toLower(extension(file.GetName())) == ".d"))

{

string fname = file.GetFullPath();

string modname = getModuleDeclarationName(fname);

if(modname.length)

return stripModule(modname);

}

}

// check sub folder

string pkgname;

for(CHierNode pNode = GetHead(); pNode; pNode = pNode.GetNext())

if(auto folder = cast(CFolderNode) pNode)

if(folder !is exclude)

{

pkgname = folder.\_GuessPackageName(false, null);

if(pkgname.length)

{

pkgname = stripPackage(pkgname, folder.GetName());

return pkgname;

}

}

// check parents

if(pkgname.empty && recurseUp)

if(auto parent = cast(CFolderNode) GetParent())

pkgname = parent.\_GuessPackageName(true, this);

if(pkgname.length)

pkgname ~= GetName() ~ ".";

return pkgname;

}

string GuessFolderPath()

{

string dir = \_GuessFolderPath(true, null);

if(dir.length)

return dir;

CProjectNode pProject = GetCVsHierarchy().GetProjectNode();

return dirName(pProject.GetFullPath());

}

string \_GuessFolderPath(bool recurseUp, CFolderNode exclude)

{

// check files in folder

for(CHierNode pNode = GetHead(); pNode; pNode = pNode.GetNext())

if(auto file = cast(CFileNode) pNode)

return dirName(pNode.GetFullPath());

for(CHierNode pNode = GetHead(); pNode; pNode = pNode.GetNext())

if(auto folder = cast(CFolderNode) pNode)

if(folder !is exclude)

{

string s = folder.\_GuessFolderPath(false, null);

if(s.length)

return dirName(s);

}

if(recurseUp)

if(auto p = cast(CFolderNode) GetParent())

{

string s = p.\_GuessFolderPath(true, this);

if(s.length)

return normalizeDir(s) ~ GetName();

}

return null;

}

// Property functions

override int GetProperty(VSHPROPID propid, out VARIANT var)

{

switch(propid)

{

case VSHPROPID\_EditLabel:

return GetEditLabel(&var.bstrVal); // can fail

default:

return super.GetProperty(propid, var);

}

}

override int SetProperty(VSHPROPID propid, in VARIANT var)

{

switch(propid)

{

case VSHPROPID\_EditLabel:

if(var.vt != VT\_BSTR)

return returnError(E\_INVALIDARG);

return SetEditLabel(var.bstrVal); // can fail

default:

return super.SetProperty(propid, var);

}

}

override int QueryStatus(

/\* [unique][in] \*/ in GUID \*pguidCmdGroup,

/\* [in] \*/ ULONG cCmds,

/\* [out][in][size\_is] \*/ OLECMD\* prgCmds,

/\* [unique][out][in] \*/ OLECMDTEXT \*pCmdText)

{

OLECMD\* Cmd = prgCmds;

HRESULT hr = S\_OK;

bool fSupported = false;

bool fEnabled = false;

bool fInvisible = false;

if(\*pguidCmdGroup == CMDSETID\_StandardCommandSet97)

{

switch(Cmd.cmdID)

{

case cmdidAddNewItem:

case cmdidAddExistingItem:

fSupported = true;

fEnabled = true;

break;

case cmdidPaste:

fSupported = true;

fEnabled = false; // ClipboardHasDropFormat();

break;

default:

hr = OLECMDERR\_E\_NOTSUPPORTED;

break;

}

}

else if (\*pguidCmdGroup == CMDSETID\_StandardCommandSet2K)

{

switch(Cmd.cmdID)

{

case cmdidExploreFolderInWindows:

fSupported = true;

string s = GuessFolderPath();

fEnabled = s.length > 0 && std.file.isDir(s);

break;

default:

hr = OLECMDERR\_E\_NOTSUPPORTED;

break;

}

}

else if(\*pguidCmdGroup == g\_commandSetCLSID)

{

switch(Cmd.cmdID)

{

case CmdNewPackage:

case CmdNewFilter:

fSupported = true;

fEnabled = true;

break;

default:

hr = OLECMDERR\_E\_NOTSUPPORTED;

break;

}

}

else

{

hr = OLECMDERR\_E\_NOTSUPPORTED;

}

if (SUCCEEDED(hr) && fSupported)

{

Cmd.cmdf = OLECMDF\_SUPPORTED;

if (fInvisible)

Cmd.cmdf |= OLECMDF\_INVISIBLE;

else if (fEnabled)

Cmd.cmdf |= OLECMDF\_ENABLED;

}

if (hr == OLECMDERR\_E\_NOTSUPPORTED)

hr = super.QueryStatus(pguidCmdGroup, cCmds, prgCmds, pCmdText);

return hr;

}

override int Exec(

/\* [unique][in] \*/ in GUID \*pguidCmdGroup,

/\* [in] \*/ DWORD nCmdID,

/\* [in] \*/ DWORD nCmdexecopt,

/\* [unique][in] \*/ in VARIANT \*pvaIn,

/\* [unique][out][in] \*/ VARIANT \*pvaOut)

{

int hr = OLECMDERR\_E\_NOTSUPPORTED;

if(\*pguidCmdGroup == CMDSETID\_StandardCommandSet97)

{

switch(nCmdID)

{

case cmdidAddNewItem:

case cmdidAddExistingItem:

hr = OnCmdAddItem(this, nCmdID == cmdidAddNewItem);

break;

default:

break;

}

}

else if (\*pguidCmdGroup == CMDSETID\_StandardCommandSet2K)

{

switch(nCmdID)

{

case cmdidExploreFolderInWindows:

hr = OnExploreFolderInWindows();

break;

case ECMD\_SHOWALLFILES:

default:

break;

}

}

else if(\*pguidCmdGroup == g\_commandSetCLSID)

{

switch(nCmdID)

{

case CmdNewPackage:

hr = OnCmdAddFolder(false);

break;

case CmdNewFilter:

hr = OnCmdAddFolder(true);

break;

default:

break;

}

}

if (hr == OLECMDERR\_E\_NOTSUPPORTED)

hr = super.Exec(pguidCmdGroup, nCmdID, nCmdexecopt, pvaIn, pvaOut);

return hr;

}

override HRESULT GetGuidProperty(VSHPROPID propid, out GUID pGuid)

{

switch (propid)

{

case VSHPROPID\_TypeGuid:

pGuid = GUID\_ItemType\_VirtualFolder;

break;

default:

return DISP\_E\_MEMBERNOTFOUND;

}

return S\_OK;

}

override uint GetContextMenu() { return IDM\_VS\_CTXT\_FOLDERNODE; }

//////////////////////////////////////////////////////////////////////

HRESULT OnCmdAddFolder(bool filter)

{

HRESULT hr = S\_OK;

// Get a reference to the project

CProjectNode pProject = GetCVsHierarchy().GetProjectNode();

// Create a new folder in the Project's folder

CFolderNode pFolder = newCom!CFolderNode;

string strThisFolder = "Filter";

if(!filter)

{

string path = GuessFolderPath();

if (path.empty)

path = dirName(pProject.GetFullPath());

strThisFolder = createNewPackageInFolder(path, "pkg");

}

pFolder.SetName(strThisFolder);

Add(pFolder);

//Fire an event to extensibility

//CAutomationEvents::FireProjectItemsEvent(pFolder, CAutomationEvents::ProjectItemsEventsDispIDs::ItemAdded);

// Since our expandable status may have changed,

// we need to refresh it in the UI

GetCVsHierarchy().OnPropertyChanged(this, VSHPROPID\_Expandable, 0);

pProject.SetProjectFileDirty(true);

// let the user rename the folder which will create the directory when finished

auto shell = ComPtr!(IVsUIShell)(queryService!(IVsUIShell));

if(shell)

{

IVsWindowFrame frame;

IVsUIHierarchyWindow uiHierarchyWindow;

scope(exit) release(frame);

scope(exit) release(uiHierarchyWindow);

VARIANT var;

hr = shell.FindToolWindow(0, &GUID\_SolutionExplorer, &frame);

if(SUCCEEDED(hr) && frame)

hr = frame.GetProperty(VSFPROPID\_DocView, &var);

if(SUCCEEDED(hr) && (var.vt == VT\_UNKNOWN || var.vt == VT\_DISPATCH))

{

uiHierarchyWindow = qi\_cast!IVsUIHierarchyWindow(var.punkVal);

var.punkVal = release(var.punkVal);

}

if(uiHierarchyWindow)

{

hr = uiHierarchyWindow.ExpandItem(GetCVsHierarchy(), pFolder.GetVsItemID(), EXPF\_SelectItem);

if(SUCCEEDED(hr))

hr = shell.PostExecCommand(&CMDSETID\_StandardCommandSet97, cmdidRename, 0, &var);

if(FAILED(hr))

hr = pFolder.OnCancelLabelEdit(); // make sure the directory is created...

}

}

return hr;

}

HRESULT OnCmdAddItem(CFolderNode folder, bool fAddNewItem, wchar\* pszSelectItem = null, wchar\* pszExpandDir = null)

{

static string strFilter = ""; // filter string (initial/final value); valid if AllowStickyFilter set

IVsAddProjectItemDlg srpAddItemDlg = queryService!(IVsAddProjectItemDlg);

if(!srpAddItemDlg)

return E\_FAIL;

scope(exit) release(srpAddItemDlg);

VSADDITEMFLAGS dwFlags;

if (fAddNewItem)

dwFlags = VSADDITEM\_AddNewItems | VSADDITEM\_SuggestTemplateName | VSADDITEM\_ShowLocationField;

else

dwFlags = VSADDITEM\_AddExistingItems | VSADDITEM\_AllowMultiSelect | VSADDITEM\_AllowStickyFilter;

string location = GetCVsHierarchy().GetProjectDir();

string folderPath = location ~ GetFolderPath(folder);

if(isExistingDir(folderPath))

location = folderPath;

auto bstrLocation = ScopedBSTR(location);

// The AddProjectItemDlg function uses and can modify the value of the filter string, so here

// we need to detach from the bstring and take the ownership of the one returned by the function.

BSTR bstrFilters = allocBSTR(strFilter);

HRESULT hr;

hr = srpAddItemDlg.AddProjectItemDlg(GetCVsHierarchy().GetVsItemID(this),

&g\_projectFactoryCLSID,

cast(IVsProject)GetCVsHierarchy(), dwFlags,

pszExpandDir, pszSelectItem,

&bstrLocation.bstr,

&bstrFilters,

null /\*&fDontShowAgain\*/);

if(bstrFilters)

{

// Take the ownership of the returned string.

strFilter = detachBSTR(bstrFilters);

}

// NOTE: AddItem() will be called via the hierarchy IVsProject to add items.

return hr;

}

HRESULT OnExploreFolderInWindows()

{

string s = GuessFolderPath();

if(s.length && std.file.isDir(s))

std.process.browse(s);

return S\_OK;

}

}

////////////////////////////////////////////////////////////////////////

class CProjectNode : CFolderNode

{

this(string filename, CVsHierarchy hierarchy)

{

mFilename = filename;

mHierarchy = hierarchy;

mTrackProjectDocuments2Helper = new CVsTrackProjectDocuments2Helper(hierarchy);

}

~this()

{

}

override uint GetContextMenu() { return IDM\_VS\_CTXT\_PROJNODE; }

override string GetFullPath()

{

return mFilename;

}

override CVsHierarchy GetCVsHierarchy()

{

return mHierarchy;

}

bool QueryEditProjectFile()

{

return true;

}

void SetProjectFileDirty(bool dirty)

{

mDirty = dirty;

}

bool IsProjectFileDirty()

{

return mDirty;

}

CVsTrackProjectDocuments2Helper GetCVsTrackProjectDocuments2Helper()

{

return mTrackProjectDocuments2Helper;

}

void SetCVsTrackProjectDocuments2Helper(CVsTrackProjectDocuments2Helper helper)

{

mTrackProjectDocuments2Helper = helper;

}

override int QueryStatus(

/\* [unique][in] \*/ in GUID \*pguidCmdGroup,

/\* [in] \*/ ULONG cCmds,

/\* [out][in][size\_is] \*/ OLECMD\* prgCmds,

/\* [unique][out][in] \*/ OLECMDTEXT \*pCmdText)

{

OLECMD\* Cmd = prgCmds;

HRESULT hr = S\_OK;

bool fSupported = false;

bool fEnabled = false;

bool fInvisible = false;

if(\*pguidCmdGroup == CMDSETID\_StandardCommandSet97)

{

switch(Cmd.cmdID)

{

case cmdidBuildSel:

case cmdidRebuildSel:

case cmdidCleanSel:

case cmdidCancelBuild:

case cmdidProjectSettings:

case cmdidBuildSln:

case cmdidUnloadProject:

case cmdidSetStartupProject:

case cmdidPropertiesWindow:

fSupported = true;

fEnabled = true;

break;

default:

hr = OLECMDERR\_E\_NOTSUPPORTED;

break;

}

}

else if(\*pguidCmdGroup == CMDSETID\_StandardCommandSet2K)

{

switch(Cmd.cmdID)

{

case cmdidBuildOnlyProject:

case cmdidRebuildOnlyProject:

case cmdidCleanOnlyProject:

case cmdidExploreFolderInWindows:

fSupported = true;

fEnabled = true;

break;

default:

hr = OLECMDERR\_E\_NOTSUPPORTED;

break;

}

}

else

{

hr = OLECMDERR\_E\_NOTSUPPORTED;

}

if (SUCCEEDED(hr) && fSupported)

{

Cmd.cmdf = OLECMDF\_SUPPORTED;

if (fInvisible)

Cmd.cmdf |= OLECMDF\_INVISIBLE;

else if (fEnabled)

Cmd.cmdf |= OLECMDF\_ENABLED;

}

if (hr == OLECMDERR\_E\_NOTSUPPORTED)

hr = super.QueryStatus(pguidCmdGroup, cCmds, prgCmds, pCmdText);

return hr;

}

override int Exec(

/\* [unique][in] \*/ in GUID \*pguidCmdGroup,

/\* [in] \*/ DWORD nCmdID,

/\* [in] \*/ DWORD nCmdexecopt,

/\* [unique][in] \*/ in VARIANT \*pvaIn,

/\* [unique][out][in] \*/ VARIANT \*pvaOut)

{

int hr = OLECMDERR\_E\_NOTSUPPORTED;

if(\*pguidCmdGroup == CMDSETID\_StandardCommandSet2K)

{

switch(nCmdID)

{

case cmdidBuildOnlyProject:

case cmdidRebuildOnlyProject:

break;

case cmdidCleanOnlyProject:

//IVsSolutionBuildManager.StartSimpleUpdateProjectConfiguration?

if(Config cfg = GetActiveConfig(GetCVsHierarchy()))

{

scope(exit) release(cfg);

if(auto win = queryService!(IVsOutputWindow)())

{

scope(exit) release(win);

IVsOutputWindowPane pane;

if(win.GetPane(&GUID\_BuildOutputWindowPane, &pane) == S\_OK)

{

scope(exit) release(pane);

cfg.StartClean(pane, 0);

}

}

}

break;

case cmdidExploreFolderInWindows:

std.process.browse(dirName(mFilename));

break;

default:

break;

}

}

if (hr == OLECMDERR\_E\_NOTSUPPORTED)

hr = super.Exec(pguidCmdGroup, nCmdID, nCmdexecopt, pvaIn, pvaOut);

return hr;

}

override int GetProperty(VSHPROPID propid, out VARIANT var)

{

switch(propid)

{

case VSHPROPID\_IsNonSearchable:

var.vt = VT\_BOOL;

var.boolVal = true;

return S\_OK;

case VSHPROPID\_BrowseObject:

return DISP\_E\_MEMBERNOTFOUND; // delegate to Project

default:

break;

}

return super.GetProperty(propid, var);

}

override int SetEditLabel(in BSTR pEditLabel)

{

string label = to\_string(pEditLabel);

SetName(label);

GetCVsHierarchy().OnPropertyChanged(this, VSHPROPID\_Name, 0);

return S\_OK;

}

private:

CVsTrackProjectDocuments2Helper mTrackProjectDocuments2Helper;

CVsHierarchy mHierarchy;

string mFilename; // always absolute

bool mDirty;

}

///////////////////////////////////////////////////////////////////////////////

abstract class CVsHierarchy :        DisposingDispatchObject,

IVsUIHierarchy,

IVsPersistHierarchyItem

{

override void Dispose()

{

m\_pParentHierarchy = release(m\_pParentHierarchy);

if(m\_pRootNode)

{

m\_pRootNode.removeFromItemMap(true);

m\_pRootNode = null;

}

}

override HRESULT QueryInterface(in IID\* riid, void\*\* pvObject)

{

if(queryInterface!(IVsHierarchy) (this, riid, pvObject))

return S\_OK;

if(queryInterface!(IVsUIHierarchy) (this, riid, pvObject))

return S\_OK;

if(queryInterface!(IVsPersistHierarchyItem) (this, riid, pvObject))

return S\_OK;

return super.QueryInterface(riid, pvObject);

}

// to be overridden

HRESULT QueryStatusSelection(in GUID \*pguidCmdGroup,

in ULONG cCmds, OLECMD \*prgCmds, OLECMDTEXT \*pCmdText,

ref CHierNode[] rgSelection, bool bIsHierCmd)

{

return returnError(OLECMDERR\_E\_NOTSUPPORTED);

}

// IVsUIHierarchy

override int QueryStatusCommand(

/\* [in] \*/ in VSITEMID itemid,

/\* [unique][in] \*/ in GUID \*pguidCmdGroup,

/\* [in] \*/ in ULONG cCmds,

/\* [size\_is][out][in] \*/ OLECMD \*prgCmds,

/\* [unique][out][in] \*/ OLECMDTEXT \*pCmdText)

{

version(none)

{

mixin(LogCallMix);

for(int i = 0; i < cCmds; i++)

//logCall(" cmd%d = (id=%d, f=%d)", i, prgCmds[i].cmdID, prgCmds[i].cmdf);

logCall("nCmdID = %s", cmd2string(\*pguidCmdGroup, prgCmds[i].cmdID));

}

CHierNode[] rgNodes = VSITEMID2Nodes(itemid);

if(rgNodes.length)

return QueryStatusSelection(pguidCmdGroup, cCmds, prgCmds, pCmdText, rgNodes, true);

return returnError(E\_NOTIMPL);

}

override int ExecCommand(

/\* [in] \*/ in VSITEMID itemid,

/\* [unique][in] \*/ in GUID \*pguidCmdGroup,

/\* [in] \*/ in DWORD nCmdID,

/\* [in] \*/ in DWORD nCmdexecopt,

/\* [unique][in] \*/ in VARIANT \*pvaIn,

/\* [unique][out][in] \*/ VARIANT \*pvaOut)

{

mixin(LogCallMix);

logCall("nCmdID = %s", cmd2string(\*pguidCmdGroup, nCmdID));

CHierNode[] rgNodes = VSITEMID2Nodes(itemid);

if (rgNodes.length == 0)

return OLECMDERR\_E\_NOTSUPPORTED;

CHierNode node = rgNodes[0];

int hr = OLECMDERR\_E\_NOTSUPPORTED;

if(\*pguidCmdGroup == GUID\_VsUIHierarchyWindowCmds)

{

switch(nCmdID)

{

case UIHWCMDID\_RightClick:

uint mnu = rgNodes.length > 1 ? GetContextMenu(rgNodes) : node.GetContextMenu();

if (mnu != IDMX\_NULLMENU)

hr = ShowContextMenu(mnu, &guidSHLMainMenu, null);

break;

case UIHWCMDID\_DoubleClick:

case UIHWCMDID\_EnterKey:

hr = node.DoDefaultAction();

break;

case UIHWCMDID\_StartLabelEdit:

hr = node.OnStartLabelEdit();

break;

case UIHWCMDID\_CommitLabelEdit:

hr = node.OnCommitLabelEdit();

break;

case UIHWCMDID\_CancelLabelEdit:

hr = node.OnCancelLabelEdit();

break;

default:

break;

}

}

if(hr == OLECMDERR\_E\_NOTSUPPORTED && node)

foreach(n; rgNodes)

if (FAILED(hr = n.Exec(pguidCmdGroup, nCmdID, nCmdexecopt, pvaIn, pvaOut)))

break;

return hr;

}

// IVsHierarchy

override int SetSite(IServiceProvider psp)

{

mixin(LogCallMix);

return returnError(E\_NOTIMPL);

}

override int GetSite(IServiceProvider \*ppSP)

{

mixin(LogCallMix);

return returnError(E\_NOTIMPL);

}

override int QueryClose(BOOL \*pfCanClose)

{

mixin(LogCallMix2);

\*pfCanClose = true;

return S\_OK;

}

override int Close()

{

mixin(LogCallMix);

return S\_OK;

}

int GetNodeIcon(CHierNode pNode)

{

if(CFileNode fnode = cast(CFileNode) pNode)

{

string tool = Config.GetStaticCompileTool(fnode, null);

switch(tool)

{

case "DMD": return kImageDSource;

case kToolResourceCompiler: return kImageResource;

case "Custom": return kImageScript;

case "None": return kImageDisabled;

default: return kImageDocument;

}

}

if(pNode == m\_pRootNode)

return kImageProject;

return kImageFolderClosed;

}

override int GetProperty(in VSITEMID itemid, in VSHPROPID propid, VARIANT\* var)

{

//mixin(LogCallMix);

CHierNode pNode = VSITEMID2Node(itemid);

if(!pNode)

return returnError(E\_INVALIDARG);

switch(propid)

{

case VSHPROPID\_EditLabel:

var.vt = VT\_BSTR;

return pNode.GetEditLabel(&var.bstrVal); // can fail

case VSHPROPID\_TypeName:

var.vt = VT\_BSTR;

var.bstrVal = allocBSTR("typename");

break;

case VSHPROPID\_ParentHierarchy:

var.vt = VT\_UNKNOWN;

var.punkVal = addref(m\_pParentHierarchy); // mProjectParent; // needs addref?

break;

case VSHPROPID\_ParentHierarchyItemid:

var.vt = VT\_I4;

var.lVal = m\_dwParentHierarchyItemid;

break;

case VSHPROPID\_Expandable:

var.vt = VT\_BOOL;

var.boolVal = pNode.Expandable();

break;

case VSHPROPID\_ExpandByDefault:

var.vt = VT\_BOOL;

var.boolVal = pNode.ExpandByDefault();

break;

case VSHPROPID\_IsHiddenItem:

var.vt = VT\_BOOL;

var.boolVal = !pNode.IsDisplayable();

break;

case VSHPROPID\_Container:

var.vt = VT\_BOOL;

var.boolVal = pNode.IsContainer();

break;

case VSHPROPID\_FirstVisibleChild:

var.vt = VT\_INT; // VT\_INT\_PTR;

var.lVal = GetFirstDisplayableNodeID(pNode);

break;

case VSHPROPID\_FirstChild:

var.vt = VT\_INT; // VT\_INT\_PTR;

var.lVal = pNode.GetFirstMemberChildID();

break;

case VSHPROPID\_NextVisibleSibling:

var.vt = VT\_INT; // VT\_INT\_PTR;

var.lVal = GetNextDisplayableNodeID(pNode);

break;

case VSHPROPID\_NextSibling:

var.vt = VT\_INT; // VT\_INT\_PTR;

var.lVal = pNode.GetNextMemberSiblingID();

break;

case VSHPROPID\_Parent:

var.vt = VT\_INT; // VT\_INT\_PTR;

var.lVal = GetVsItemID(pNode.GetParent());

break;

case VSHPROPID\_Root:

var.vt = VT\_INT; // VT\_INT\_PTR;

var.lVal = VSITEMID\_ROOT;

break;

case VSHPROPID\_IconImgList:

var.vt = VT\_I4;

auto himagelst = LoadImageList(g\_hInst, MAKEINTRESOURCEA(BMP\_DIMAGELIST), 16, 16);

var.lVal = cast(int) himagelst;

break;

case VSHPROPID\_IconHandle:

case VSHPROPID\_IconIndex:

var.vt = VT\_I4;

var.lVal = GetNodeIcon(pNode);

break;

case VSHPROPID\_OpenFolderIconIndex:

var.vt = VT\_I4;

var.lVal = pNode == m\_pRootNode ? kImageProject : kImageFolderOpened;

break;

case VSHPROPID\_IsNonLocalStorage:

case VSHPROPID\_HandlesOwnReload:

case VSHPROPID\_CanBuildFromMemory:

var.vt = VT\_BOOL;

var.boolVal = false;

break;

case VSHPROPID\_DefaultEnableDeployProjectCfg:

case VSHPROPID\_DefaultEnableBuildProjectCfg:

var.vt = VT\_BOOL;

var.boolVal = true;

break;

/+

case VSHPROPID\_ExtObject:

var.vt = VT\_DISPATCH;

var.pdispVal = addref(mExtProject);

break;

//return DISP\_E\_MEMBERNOTFOUND;

+/

case VSHPROPID\_BrowseObject:

//var.vt = VT\_UNKNOWN;

//var.punkVal = null;

//break;

case VSHPROPID\_ProjectDir:

// ReloadableProjectFile, IsNonLocalStorage, CanBuildFromMemory,

// DefaultEnableBuildProjectCfg, DefaultEnableDeployProjectCfg,

// IsNonSearchable, HasEnumerationSideEffects, ExtObject

// 1001

//case VSHPROPID2.EnableDataSourceWindow:

//case VSHPROPID2.DebuggeeProcessId:

case cast(VSHPROPID) 1001:

default:

if(pNode.GetProperty(propid, \*var) == S\_OK)

break;

//logCall("Getting unknown property %d for item %x!", propid, itemid);

return DISP\_E\_MEMBERNOTFOUND;

// return returnError(E\_NOTIMPL); // DISP\_E\_MEMBERNOTFOUND;

}

return S\_OK;

}

override int SetProperty(in VSITEMID itemid, in VSHPROPID propid, in VARIANT var)

{

CHierNode pNode = VSITEMID2Node(itemid);

if(!pNode)

return returnError(E\_INVALIDARG);

HRESULT hr = pNode.SetProperty(propid, var);

if(hr != DISP\_E\_MEMBERNOTFOUND && hr != E\_NOTIMPL)

return hr;

switch(propid)

{

case VSHPROPID\_ParentHierarchy:

if(var.vt != VT\_UNKNOWN)

return returnError(E\_INVALIDARG);

m\_pParentHierarchy = release(m\_pParentHierarchy);

m\_pParentHierarchy = addref(cast(IUnknown)var.punkVal);

break;

case VSHPROPID\_ParentHierarchyItemid:

if(var.vt != VT\_I4)

return returnError(E\_INVALIDARG);

m\_dwParentHierarchyItemid = var.lVal;

break;

default:

logCall("Setting unknown property %d for item %x!", propid, itemid);

return DISP\_E\_MEMBERNOTFOUND;

}

return S\_OK;

}

override int GetGuidProperty(in VSITEMID itemid, in VSHPROPID propid, GUID\* pGuid)

{

if(CHierNode pNode = VSITEMID2Node(itemid))

return pNode.GetGuidProperty(propid, \*pGuid);

return returnError(E\_INVALIDARG);

}

override int GetNestedHierarchy(in VSITEMID itemid, in IID\* iidHierarchyNested, void \*\*ppHierarchyNested, VSITEMID\* pitemidNested)

{

mixin(LogCallMix);

if(CHierNode pNode = VSITEMID2Node(itemid))

return pNode.GetNestedHierarchy(iidHierarchyNested, ppHierarchyNested, \*pitemidNested);

return returnError(E\_FAIL);

}

override int GetCanonicalName(in VSITEMID itemid, BSTR \*pbstrName)

{

logCall("GetCanonicalName(this=%s, itemid=%s, pbstrMkDocument=%s)", cast(void\*)this, \_toLog(itemid), \_toLog(pbstrName));

scope(exit)

logCall(" GetCanonicalName return %s", \_toLog(\*pbstrName));

if(CHierNode pNode = VSITEMID2Node(itemid))

{

\*pbstrName = allocBSTR(pNode.GetCanonicalName());

return S\_OK;

}

return returnError(E\_INVALIDARG);

}

override int ParseCanonicalName(in wchar\* pszName, VSITEMID\* pitemid)

{

mixin(LogCallMix2);

string docName = toLower(to\_string(pszName));

CHierNode node = searchNode(GetRootNode(), delegate (CHierNode n) { return n.GetCanonicalName() == docName; });

\*pitemid = GetVsItemID(node);

return node ? S\_OK : E\_FAIL;

}

override int Unused0()

{

mixin(LogCallMix);

return returnError(E\_NOTIMPL);

}

override int AdviseHierarchyEvents(IVsHierarchyEvents pEventSink, uint \*pdwCookie)

{

mixin(LogCallMix);

mLastHierarchyEventSinkCookie++;

mHierarchyEventSinks[mLastHierarchyEventSinkCookie] = addref(pEventSink);

\*pdwCookie = mLastHierarchyEventSinkCookie;

return S\_OK;

}

override int UnadviseHierarchyEvents(in uint dwCookie)

{

//                mixin(LogCallMix);

if(dwCookie in mHierarchyEventSinks)

{

release(mHierarchyEventSinks[dwCookie]);

mHierarchyEventSinks.remove(dwCookie);

return S\_OK;

}

return returnError(E\_INVALIDARG);

}

override int Unused1()

{

mixin(LogCallMix);

return returnError(E\_NOTIMPL);

}

override int Unused2()

{

mixin(LogCallMix);

return returnError(E\_NOTIMPL);

}

override int Unused3()

{

mixin(LogCallMix);

return returnError(E\_NOTIMPL);

}

override int Unused4()

{

mixin(LogCallMix);

return returnError(E\_NOTIMPL);

}

// IVsPersistHierarchyItem

override int IsItemDirty(

/\* [in] \*/ in VSITEMID itemid,

/\* [in] \*/ IUnknown punkDocData,

/\* [out] \*/ BOOL \*pfDirty)

{

auto srpPersistDocData = ComPtr!(IVsPersistDocData)(punkDocData);

if(!srpPersistDocData)

return E\_INVALIDARG;

return srpPersistDocData.IsDocDataDirty(pfDirty);

}

override int SaveItem(

/\* [in] \*/ in VSSAVEFLAGS dwSave,

/\* [in] \*/ in wchar\* pszSilentSaveAsName,

/\* [in] \*/ in VSITEMID itemid,

/\* [in] \*/ IUnknown punkDocData,

/\* [out] \*/ BOOL\* pfCanceled)

{

// validate itemid.

if (itemid == VSITEMID\_ROOT || itemid == VSITEMID\_SELECTION || !VSITEMID2Node(itemid))

return E\_INVALIDARG;

if (!punkDocData)

return OLE\_E\_NOTRUNNING; // we can only perform save if the document is open

BSTR bstrMkDocumentNew;

HRESULT hr = E\_FAIL;

if (VSSAVE\_SilentSave & dwSave)

{

auto srpFileFormat = ComPtr!(IPersistFileFormat)(punkDocData);

auto pIVsUIShell = ComPtr!(IVsUIShell)(queryService!(IVsUIShell));

if(srpFileFormat && pIVsUIShell)

hr = pIVsUIShell.SaveDocDataToFile(dwSave, srpFileFormat, pszSilentSaveAsName, &bstrMkDocumentNew, pfCanceled);

}

else

{

auto srpPersistDocData = ComPtr!(IVsPersistDocData)(punkDocData);

if(srpPersistDocData)

hr = srpPersistDocData.SaveDocData(dwSave, &bstrMkDocumentNew, pfCanceled);

}

freeBSTR(bstrMkDocumentNew); // release string

// if a SaveAs occurred we need to update to the fact our item's name has changed.

// this includes the following:

// 1. call RenameDocument on the RunningDocumentTable

// 2. update the full path name for the item in our hierarchy

// 3. a directory-based project may need to transfer the open editor to the

// MiscFiles project if the new file is saved outside of the project directory.

// This is accomplished by calling IVsExternalFilesManager::TransferDocument

// This work can not be done by CVsHierarchy::SaveItem; this must be done in a

// derived subclass implementation of OnHandleSaveItemRename.

//if ((!\*pfCanceled) && bstrMkDocumentNew != NULL)

//        hr = OnHandleSaveItemRename(itemid, punkDocData, bstrMkDocumentNew);

return hr;

}

///////////////////////////////////////////////////////////////

CHierNode VSITEMID2Node(VSITEMID itemid)

{

switch (itemid)

{

case VSITEMID\_NIL:

assert(\_false, "error: known invalid VSITEMID");

return null;

case VSITEMID\_ROOT:

return GetRootNode();

case VSITEMID\_SELECTION:

assert(\_false, "error: Hierarchy illegaly called with VSITEMID\_SELECTION");

return null;

default:

synchronized(gVsItemMap\_sync)

if(CHierNode\* pNode = itemid in gVsItemMap)

if(pNode.GetRootNode() == GetRootNode())

return \*pNode;

}

return null;

}

///////////////////////////////////////////////////////////////

CHierNode[] VSITEMID2Nodes(VSITEMID itemid)

{

CHierNode[] nodes;

switch (itemid)

{

case VSITEMID\_NIL:

break;

case VSITEMID\_ROOT:

nodes ~= GetRootNode();

break;

case VSITEMID\_SELECTION:

GetSelectedNodes(nodes);

break;

default:

synchronized(gVsItemMap\_sync)

if(CHierNode\* pNode = itemid in gVsItemMap)

nodes ~= \*pNode;

}

return nodes;

}

// Virtuals called in response to VSHPROPID\_FirstChild, VSHPROPID\_GextNextSibling. Defaults

// just call pNode's GetFirstChild()/GetNext() methods. Override to display the nodes differently

VSITEMID GetFirstDisplayableNodeID(CHierNode pNode)

{

return pNode.GetFirstChildID(true);

}

VSITEMID GetNextDisplayableNodeID(CHierNode pNode)

{

return GetVsItemID(pNode.GetNext());

}

// Following function returns the previous node in the hierwindow. It is obviously dependant on

// the sorting way GetFirstDisplayableNode, GetNextDisplayable node are implemented.

CHierNode GetPrevDisplayableNode(CHierNode pNode)

{

assert(pNode.IsDisplayable());

return pNode.GetParent().GetPrevChildOf(pNode);

}

public: // IVsHierarchyEvent propagation

HRESULT OnItemAdded(CHierNode pNodeParent, CHierNode pNodePrev, CHierNode pNodeAdded)

{

GetProjectNode().SetProjectFileDirty(true);

assert(pNodeParent && pNodeAdded);

VSITEMID itemidParent = GetVsItemID(pNodeParent);

VSITEMID itemidSiblingPrev = GetVsItemID(pNodePrev);

VSITEMID itemidAdded = GetVsItemID(pNodeAdded);

foreach (advise; mHierarchyEventSinks)

advise.OnItemAdded(itemidParent, itemidSiblingPrev, itemidAdded);

return S\_OK;

}

HRESULT OnItemDeleted(CHierNode pNode)

{

GetProjectNode().SetProjectFileDirty(true);

VSITEMID itemid = GetVsItemID(pNode);

// Note that in some cases (deletion of project node for example), an Advise

// may be removed while we are iterating over it. To get around this problem we

// take a snapshot of the advise list and walk that.

IVsHierarchyEvents[] sinks;

foreach (advise; mHierarchyEventSinks)

sinks ~= advise;

foreach (advise; sinks)

advise.OnItemDeleted(itemid);

return S\_OK;

}

HRESULT OnPropertyChanged(CHierNode pNode, VSHPROPID propid, DWORD flags)

{

GetProjectNode().SetProjectFileDirty(true);

VSITEMID itemid = GetVsItemID(pNode);

if (pNode.IsDisplayable())

foreach (advise; mHierarchyEventSinks)

advise.OnPropertyChanged(itemid, propid, flags);

return S\_OK;

}

HRESULT OnInvalidateItems(CHierNode pNode)

{

VSITEMID itemid = GetVsItemID(pNode);

foreach (advise; mHierarchyEventSinks)

advise.OnInvalidateItems(itemid);

return S\_OK;

}

HRESULT OnInvalidateIcon(HICON hIcon)

{

foreach (advise; mHierarchyEventSinks)

advise.OnInvalidateIcon(hIcon);

return S\_OK;

}

string GetProjectDir() { return dirName(m\_pRootNode.GetFullPath()); }

CProjectNode GetProjectNode() { return m\_pRootNode; }

CHierContainer GetRootNode() { return m\_pRootNode; }

void SetRootNode(CProjectNode root) { m\_pRootNode = root; }

VSITEMID GetVsItemID(CHierNode node)

{

if(!node)

return VSITEMID\_NIL;

if(node is GetRootNode())

return VSITEMID\_ROOT;

return node.GetVsItemID();

}

IServiceProvider getServiceProvider()

{

return null;

}

IVsHierarchy GetIVsHierarchy()

{

return this;

}

//---------------------------------------------------------------------------

// fill out an array of selected nodes

//---------------------------------------------------------------------------

HRESULT GetSelectedNodes(ref CHierNode[] rgNodes)

{

IVsMonitorSelection srpMonSel = queryService!(IVsMonitorSelection);

if(!srpMonSel)

return returnError(E\_FAIL);

HRESULT hr = S\_OK;

VSITEMID itemid; // if VSITEMID\_SELECTION then multiselection

CHierNode pNode = null;

IVsHierarchy srpIVsHierarchy; // if NULL then selection spans VsHierarchies

IVsMultiItemSelect srpIVsMultiItemSelect;

ISelectionContainer srpISelectionContainer; // unused?

hr = srpMonSel.GetCurrentSelection(&srpIVsHierarchy, &itemid, &srpIVsMultiItemSelect, &srpISelectionContainer);

if(hr == S\_OK)

{

if (VSITEMID\_NIL == itemid)

{ // nothing selected

}

else if (VSITEMID\_SELECTION != itemid)

{        // Single selection. Note that callers of this function, may try to get the

// selection when we aren't the active hierarchy - for this reason we need

// to validate that the selected item belongs to us.

if(srpIVsHierarchy is GetIVsHierarchy())

{

pNode = VSITEMID2Node(itemid);

if (pNode)

rgNodes ~= pNode;

else

logCall(" ERROR: invalid VSITEMID in selection");

}

}

else if (srpIVsMultiItemSelect)

{

ULONG cItems = 0;

BOOL fSingleHierarchy = TRUE;

hr = srpIVsMultiItemSelect.GetSelectionInfo(&cItems, &fSingleHierarchy);

if (SUCCEEDED(hr))

{

assert(0 < cItems); // nothing selected should already be filtered out

if(!fSingleHierarchy || srpIVsHierarchy is GetIVsHierarchy())

{

VSITEMSELECTION[] pItemSel = new VSITEMSELECTION[cItems];

VSGSIFLAGS fFlags = fSingleHierarchy ? GSI\_fOmitHierPtrs : cast(VSGSIFLAGS) 0;

hr = srpIVsMultiItemSelect.GetSelectedItems(fFlags, cItems, pItemSel.ptr);

if (SUCCEEDED(hr))

{

ULONG i;

for (i = 0; i < cItems; ++i)

{

if (fSingleHierarchy || pItemSel[i].pHier is GetIVsHierarchy())

{

pNode = VSITEMID2Node(pItemSel[i].itemid);

assert(pNode); // why is there an invalid itemid?

if (pNode)

rgNodes ~= pNode;

}

}

if (!fSingleHierarchy)

{ // release all the hierarchies

for (i = 0; i < cItems; ++i)

release(pItemSel[i].pHier);

}

}

}

}

}

}

release(srpMonSel);

release(srpIVsHierarchy);

release(srpIVsMultiItemSelect);

release(srpISelectionContainer);

return hr;

}

uint GetContextMenu(CHierNode[] rgSelection)

{

bool IsItemNodeCtx(uint idmx)

{

return (idmx == IDM\_VS\_CTXT\_ITEMNODE || idmx == IDM\_VS\_CTXT\_XPROJ\_MULTIITEM);

}

uint idmxMenu = IDMX\_NULLMENU;

bool fProjSelected = false;

foreach(pNode; rgSelection)

{

uint idmxTemp = pNode.GetContextMenu();

if(idmxTemp == IDMX\_NULLMENU)

{ // selection contains node that does not have a ctx menu

idmxMenu = IDMX\_NULLMENU;

break;

}

else if(IDM\_VS\_CTXT\_PROJNODE == idmxTemp)

{

// selection includes project node

fProjSelected = TRUE;

}

else if (idmxMenu == IDMX\_NULLMENU || idmxMenu == idmxTemp)

{ // homogeneous selection

idmxMenu = idmxTemp;

}

else if (IsItemNodeCtx(idmxTemp) && IsItemNodeCtx(idmxMenu))

{

// heterogeneous set of nodes that support common node commands

idmxMenu = IDM\_VS\_CTXT\_XPROJ\_MULTIITEM;

}

else

{ // heterogeneous set of nodes that have no common commands

idmxMenu = IDMX\_NULLMENU;

break;

}

}

// Multi-selection involving project node.

if (idmxMenu != IDMX\_NULLMENU && fProjSelected)

idmxMenu = IDM\_VS\_CTXT\_XPROJ\_PROJITEM;

return idmxMenu;

}

void SetErrorInfo(HRESULT hr, string txt)

{

auto srpUIManager = queryService!(IVsUIShell);

if(!srpUIManager)

return;

scope(exit) release(srpUIManager);

auto wtxt = \_toUTF16z(txt);

wchar\* wEmptyString = cast(wchar\*) "\0"w.ptr;

srpUIManager.SetErrorInfo(hr, wtxt, 0, wEmptyString, wEmptyString);

}

HRESULT OpenDoc(CFileNode pNode,

/\* [in] \*/ bool fNewFile /\*= FALSE\*/,

/\* [in] \*/ bool fUseOpenWith /\*= FALSE\*/,

/\* [in] \*/ bool fShow /\*= TRUE \*/,

/\* [in] \*/ in GUID\* rguidLogicalView = &LOGVIEWID\_Primary,

/\* [in] \*/ in GUID\* rguidEditorType = &GUID\_NULL,

/\* [in] \*/ in wchar\* pszPhysicalView = null,

/\* [in] \*/ IUnknown punkDocDataExisting = DOCDATAEXISTING\_UNKNOWN,

/\* [out] \*/ IVsWindowFrame\* ppWindowFrame = null)

{

HRESULT hr = S\_OK;

// Get the IVsUIShellOpenDocument service so we can ask it to open a doc window

IVsUIShellOpenDocument pIVsUIShellOpenDocument = queryService!(IVsUIShellOpenDocument);

if(!pIVsUIShellOpenDocument)

return returnError(E\_FAIL);

scope(exit) release(pIVsUIShellOpenDocument);

string strFullPath = pNode.GetFullPath();

auto wstrFullPath = \_toUTF16z(strFullPath);

// do not force file to belong to only one project

VSITEMID itemid = GetVsItemID(pNode);

IVsUIHierarchy pHier = this;

IVsUIHierarchy hierOpen;

VSITEMID itemidOpen;

IVsWindowFrame windowFrame;

BOOL fOpen;

scope(exit) release(windowFrame);

scope(exit) release(hierOpen);

hr = pIVsUIShellOpenDocument.IsDocumentOpen(null, 0, wstrFullPath, rguidLogicalView,

IDO\_ActivateIfOpen,

&hierOpen, &itemidOpen, &windowFrame, &fOpen);

if(SUCCEEDED(hr) && fOpen)

return hr;

if(!pszPhysicalView)

{

VSOSEFLAGS openFlags = OSE\_ChooseBestStdEditor;

if(fUseOpenWith)

openFlags = OSE\_UseOpenWithDialog;

if(fNewFile)

openFlags |= OSE\_OpenAsNewFile;

hr = pIVsUIShellOpenDocument.OpenStandardEditor(

/\* [in] VSOSEFLAGS grfOpenStandard \*/ openFlags,

/\* [in] LPCOLESTR pszMkDocument \*/ wstrFullPath,

/\* [in] REFGUID rguidLogicalView \*/ rguidLogicalView,

/\* [in] LPCOLESTR pszOwnerCaption \*/ \_toUTF16z("%3"),

/\* [in] IVsUIHierarchy \*pHier \*/ pHier,

/\* [in] VSITEMID itemid \*/ itemid,

/\* [in] IUnknown \*punkDocDataExisting \*/ punkDocDataExisting,

/\* [in] IServiceProvider \*pSP \*/ null,

/\* [out, retval] IVsWindowFrame \*\*ppWindowFrame \*/ &windowFrame);

}

else

{

VSOSPEFLAGS openFlags = fNewFile ? OSPE\_OpenAsNewFile : cast(VSOSPEFLAGS) 0;

hr = pIVsUIShellOpenDocument.OpenSpecificEditor(

/\* VSOSPEFLAGS grfOpenSpecific \*/ openFlags,

/\* LPCOLESTR pszMkDocument \*/ wstrFullPath,

/\* REFGUID rguidEditorType \*/ rguidEditorType,

/\* LPCOLESTR pszPhysicalView \*/ cast(wchar\*) pszPhysicalView,

/\* REFGUID rguidLogicalView \*/ rguidLogicalView,

/\* LPCOLESTR pszOwnerCaption \*/ \_toUTF16z("%3"),

/\* IVsUIHierarchy \*pHier \*/ pHier,

/\* VSITEMID itemid \*/ itemid,

/\* IUnknown \*punkDocDataExisting \*/ punkDocDataExisting,

/\* IServiceProvider \*pSPHierContext \*/ null,

/\* IVsWindowFrame \*\*ppWindowFrame \*/ &windowFrame);

}

// Note that for external editors we don't get an windowFrame.

if(SUCCEEDED(hr) && windowFrame)

{

if(fNewFile)

{

// SetUntitledDocPath is called by all projects after a new document instance is created.

// Editors use the same CreateInstance/InitNew design pattern of standard COM objects.

// Editors can use this method to perform one time initializations that are required after a new

// document instance was created via IVsEditorFactory::CreateEditorInstance(CEF\_CLONEFILE,...).

// NOTE: Ideally this method would be called InitializeNewDocData but it is too late to rename this method.

// Most editors can ignore the parameter passed. It is a legacy of historical insignificance.

VARIANT var;

HRESULT hrTemp = windowFrame.GetProperty(VSFPROPID\_DocData, &var);

if(SUCCEEDED(hrTemp) && var.vt == VT\_UNKNOWN && var.punkVal)

{

IVsPersistDocData srpDocData;

hrTemp = var.punkVal.QueryInterface(&IVsPersistDocData.iid, cast(void\*\*)&srpDocData);

if(SUCCEEDED(hrTemp) && srpDocData)

{

srpDocData.SetUntitledDocPath(wstrFullPath);

release(srpDocData);

}

}

}

// Show window

if (fShow)

windowFrame.Show();

// Return window frame if requested

if(ppWindowFrame)

\*ppWindowFrame = addref(windowFrame);

}

return hr;

}

HRESULT AddItemSpecific(CHierContainer pNode,

/\* [in] \*/ VSADDITEMOPERATION dwAddItemOperation,

/\* [in] \*/ in wchar\* pszItemName,

/\* [in] \*/ uint cFilesToOpen,

/\* [in, size\_is(cFilesToOpen)] \*/ in wchar\*\* rgpszFilesToOpen,

/\* [in] \*/ in HWND hwndDlg,

/\* [in] \*/ VSSPECIFICEDITORFLAGS grfEditorFlags,

/\* [in] \*/ in GUID\* rguidEditorType,

/\* [in] \*/ in wchar\* pszPhysicalView,

/\* [in] \*/ in GUID\* rguidLogicalView,

/\* [in] \*/ bool moveIfInProject,

/\* [out, retval] \*/ VSADDRESULT\* pResult)

{

\*pResult = ADDRESULT\_Failure;

HRESULT hr = S\_OK;

HRESULT hrTemp = S\_OK;

CProjectNode pProject = GetProjectNode();

// CExecution singleEx(&GetExecutionCtx());

// Return if the project file is not editable or the project file was reloaded

if(!pProject.QueryEditProjectFile())

return OLE\_E\_PROMPTSAVECANCELLED;

switch(dwAddItemOperation)

{

case VSADDITEMOP\_LINKTOFILE:

// because we are a reference-based project system our handling for

// LINKTOFILE is the same as OPENFILE.

// a storage-based project system which handles OPENFILE by copying

// the file into the project directory would have distinct handling

// for LINKTOFILE vs. OPENFILE.

// we fall through to VSADDITEMOP\_OPENFILE....

case VSADDITEMOP\_OPENFILE:

case VSADDITEMOP\_CLONEFILE:

{

bool fNewFile = (dwAddItemOperation == VSADDITEMOP\_CLONEFILE);

for(uint i = 0; i < cFilesToOpen; i++)

{

CHierNode pNewNode;

if (fNewFile)

{

assert(cFilesToOpen == 1);

assert(rgpszFilesToOpen[i]);

assert(pszItemName);

pNewNode = AddNewNode(pNode, to\_string(rgpszFilesToOpen[i]), to\_string(pszItemName));

}

else

{

// create and add node for the existing file to the project

pNewNode = AddExistingFile(pNode, to\_string(rgpszFilesToOpen[i]), false, false, moveIfInProject);

}

if(!pNewNode)

{

// This means that we return an error code if even one

// of the Items failed to Add (in the add existing files case)

hr = E\_FAIL;

continue;

}

CFileNode pFileNode = cast(CFileNode) pNewNode;

// we are not opening an existing file if an editor is not specified

if (!fNewFile && \*rguidEditorType == GUID\_NULL)

continue;

if(!pFileNode)

continue;

// open the item

assert(grfEditorFlags & VSSPECIFICEDITOR\_DoOpen);

IVsWindowFrame srpWindowFrame;

bool useView = (grfEditorFlags & VSSPECIFICEDITOR\_UseView) != 0;

// Standard open file

hrTemp = OpenDoc(pFileNode, fNewFile /\*fNewFile\*/,

false /\*fUseOpenWith\*/,

true /\*fShow\*/,

rguidLogicalView,

rguidEditorType,

useView ? null : pszPhysicalView,

null,

&srpWindowFrame);

if (FAILED(hrTemp))

{

// These don't affect the return value of this function because

// by this stage the file has been sucessfully added to the project.

// But the problem can be reported to the user.

}

}

break;

}

case VSADDITEMOP\_RUNWIZARD: // Wizard was selected

return RunWizard(pNode,

/\* [in] LPCOLESTR pszItemName \*/ pszItemName,

/\* [in] ULONG cFilesToOpen \*/ cFilesToOpen,

/\* [in] LPCOLESTR rgpszFilesToOpen[]\*/ rgpszFilesToOpen,

/\* [in] HWND hwndDlg \*/ hwndDlg,

/\* [out] VSADDRESULT \* pResult \*/ pResult);

default:

\*pResult = ADDRESULT\_Failure;

hr = E\_INVALIDARG;

}

if (SUCCEEDED(hr))

\*pResult = ADDRESULT\_Success;

/+

if(GetExecutionCtx().IsCancelled() || hr == E\_ABORT || hr == OLE\_E\_PROMPTSAVECANCELLED)

{

\*pResult = ADDRESULT\_Cancel;

hr = S\_OK;

}

+/

return hr;

}

CHierNode AddNewNode(CHierContainer pNode, string strFullPathSource, string strNewFileName)

{

HRESULT hr = S\_OK;

if(!CheckFileName(strNewFileName))

{

SetErrorInfo(E\_FAIL, format("The filename is not valid: %s", strNewFileName));

return null;

}

if(!isAbsolute(strNewFileName))

strNewFileName = GetProjectDir() ~ "\\" ~ strNewFileName;

bool dir = isExistingDir(strFullPathSource);

// If target != source then we need to copy

if (CompareFilenames(strFullPathSource, strNewFileName) != 0)

{

bool fCopied = true;

bool bStatus = false;

// Don't force an overwrite.

if(std.file.exists(strNewFileName))

{

string msg = format("%s already exists. Overwrite?", strNewFileName);

string caption = "Add new file";

int msgRet = UtilMessageBox(msg, MB\_YESNOCANCEL | MB\_ICONEXCLAMATION, caption);

if (msgRet != IDYES)

return null;

string docName = toLower(strNewFileName);

CHierNode node = searchNode(GetRootNode(), delegate (CHierNode n) { return n.GetCanonicalName() == docName; });

// Remove the corresponding node from the hierarchy, we will add a new one with the same name below

if(node)

hr = node.GetParent().Delete(node, this);

assert(SUCCEEDED(hr));

}

try

{

if(dir)

std.file.mkdir(strNewFileName);

else

{

string txt = cast(string) std.file.read(strFullPathSource);

string modname = safeFilename(stripExtension(baseName(strNewFileName)));

txt = replace(txt, "$safeitemname$", modname);

if(txt.indexOf("$modulename$") >= 0)

{

string pkg;

if(auto folder = cast(CFolderNode) pNode)

pkg = folder.GuessPackageName();

if(pkg.length)

modname = pkg ~ "." ~ modname;

txt = replace(txt, "$modulename$", modname);

}

std.file.write(strNewFileName, txt);

}

}

catch(Exception e)

{

// get windows error and produce error info

writeToBuildOutputPane(e.msg);

return null;

}

// template was read-only, but our file should not be

//if (fCopied)

//        SetFileAttributes(strNewFileName, FILE\_ATTRIBUTE\_ARCHIVE);

}

if(dir)

{

CFolderNode pFolder = newCom!CFolderNode;

string strThisFolder = baseName(strNewFileName);

pFolder.SetName(strThisFolder);

pNode.Add(pFolder);

return pFolder;

}

// Now that we have made a copy of the template file, let's add our new file to the project

return AddExistingFile(pNode, strNewFileName);

}

CHierNode AddExistingFile(CHierContainer pNode, string strFullPathSource,

bool fSilent = false, bool fLoad = false, bool moveIfInProject = false)

{

// get the proper file name

string strFullPath = strFullPathSource;

if(!CheckFileName(strFullPath))

return null;

bool dir = false;

// check the file specified if we are not merely opening an existing project

if (!fLoad)

{

if(!std.file.exists(strFullPath))

{

if (!fSilent)

{

string msg = format("%s does not exist.", strFullPath);

UtilMessageBox(msg, MB\_OK, "Add file");

}

return null;

}

if(std.file.isDir(strFullPath))

{

dir = true;

}

else

{

string canonicalName = toLower(strFullPath);

CHierNode node = searchNode(GetRootNode(), delegate (CHierNode n) { return n.GetCanonicalName() == canonicalName; });

if(node && !moveIfInProject)

{

if (!fSilent)

{

string msg = format("%s is already in the project.", strFullPath);

UtilMessageBox(msg, MB\_OK, "Add file");

}

return null;

}

}

}

// the file looks ok

CProjectNode pProject = GetProjectNode();

CVsTrackProjectDocuments2Helper pTrackDoc = pProject.GetCVsTrackProjectDocuments2Helper();

if (!fSilent)

{

if(dir)

{

string bname = baseName(strFullPath);

for(CHierNode node = pNode.GetHeadEx(true); node; node = node.GetNext(true))

if(toLower(bname) == node.GetName())

{

if (!fSilent)

{

string msg = format("%s already exists in folder.", bname);

UtilMessageBox(msg, MB\_OK, "Add file");

}

return null;

}

}

else if(!pTrackDoc.CanAddItem(strFullPath))

return null;

}

string projDir = GetProjectDir();

CHierNode pNewNode;

if(dir)

{

pNewNode = newCom!CFolderNode(baseName(strFullPath));

}

else

{

string relPath = makeRelative(strFullPath, projDir);

pNewNode = newCom!CFileNode(relPath);

}

pNode.Add(pNewNode);

if (!fSilent)

{

pTrackDoc.OnItemAdded(pNewNode);

//Fire an event to extensibility

//CAutomationEvents::FireProjectItemsEvent(pNewFile, CAutomationEvents::ProjectItemsEventsDispIDs::ItemAdded);

}

pProject.GetCVsHierarchy().OnPropertyChanged(pNode, VSHPROPID\_Expandable, 0);

pProject.SetProjectFileDirty(true);

if(dir && !fLoad && !moveIfInProject)

{

CHierContainer cont = cast(CHierContainer) pNewNode;

assert(cont);

foreach(string fname; dirEntries(strFullPath, SpanMode.shallow))

if(!startsWith(baseName(fname), "."))

if(!AddExistingFile(cont, fname, fSilent))

return null;

}

return pNewNode;

}

HRESULT RunWizard(CHierContainer pNode,

/\* [in] \*/ in wchar\* pszItemName,

/\* [in] \*/ ULONG cFilesToOpen,

/\* [in, size\_is(cFilesToOpen)] \*/ in wchar\*\* rgpszFilesToOpen,

/\* [in] \*/ in HWND hwndDlg,

/\* [out, retval] \*/ VSADDRESULT\* pResult)

{

if(cFilesToOpen < 1)

return E\_FAIL;

string itemName = to\_string(pszItemName);

string vszFile = to\_string(rgpszFilesToOpen[0]);

if(icmp(baseName(vszFile), "package.vsz") == 0)

{

\*pResult = ADDRESULT\_Failure;

try

{

mkdir(itemName);

if(AddExistingFile(pNode, itemName))

\*pResult = ADDRESULT\_Success;

}

catch(Exception)

{

}

return S\_OK;

}

return E\_NOTIMPL;

}

protected:

CProjectNode m\_pRootNode;

// Hierarchy event advises

IVsHierarchyEvents[uint] mHierarchyEventSinks;

uint mLastHierarchyEventSinkCookie;

BOOL m\_fHierClosed;

// Properties to support being used as a nested hierarchy

IUnknown m\_pParentHierarchy;

VSITEMID m\_dwParentHierarchyItemid;

// support VSHPROPID\_OwnerKey

wstring m\_bstrOwnerKey;

static BOOL g\_bStartedDrag;

static BOOL g\_bInContextMenu; // is OK to support Cut/Copy,Paste/Rename/etc.

}

// This file is part of Visual D

//

// Visual D integrates the D programming language into Visual Studio

// Copyright (c) 2010 by Rainer Schuetze, All Rights Reserved

//

// Distributed under the Boost Software License, Version 1.0.

// See accompanying file LICENSE\_1\_0.txt or copy at <http://www.boost.org/LICENSE_1_0.txt>

module visuald.hierutil;

import visuald.windows;

import std.string;

import std.file;

import std.path;

import std.utf;

import std.array;

import std.conv;

import core.stdc.wchar\_ : wcslen;

import stdext.path;

import stdext.array;

import stdext.string;

import sdk.port.vsi;

import sdk.vsi.vsshell;

import sdk.vsi.objext;

import sdk.vsi.uilocale;

import dte = sdk.vsi.dte80a;

import dte2 = sdk.vsi.dte80;

import visuald.comutil;

import visuald.fileutil;

import visuald.logutil;

import visuald.stringutil;

import visuald.dpackage;

import visuald.dproject;

import visuald.completion;

import visuald.chiernode;

import visuald.chiercontainer;

import visuald.hierarchy;

import visuald.config;

import visuald.winctrl;

const uint \_MAX\_PATH = 260;

///////////////////////////////////////////////////////////////////////

int CompareFilenames(string f1, string f2)

{

return icmp(f1, f2);

/+

if(f1 == f2)

return 0;

if(f1 < f2)

return -1;

return 1;

+/

}

bool ContainsInvalidFileChars(string name)

{

string invalid = "\\/:\*?\"<>|";

foreach(dchar ch; name)

if(indexOf(invalid, ch) >= 0)

return true;

return false;

}

bool CheckFileName(string fileName)

{

if (fileName.length == 0 || fileName.length >= \_MAX\_PATH)

return false;

string base = baseName(fileName);

if(base.length == 0)

return false;

if(ContainsInvalidFileChars(base))

return false;

base = getNameWithoutExt(base);

if(base.length == 0)

return true; // file starts with '.'

static string[] reservedNames =

[

"CON", "PRN", "AUX", "CLOCK$", "NUL",

"COM1","COM2", "COM3","COM4","COM5", "COM6", "COM7","COM8", "COM9",

"LPT1","LPT2", "LPT3","LPT4","LPT5", "LPT6", "LPT7","LPT8", "LPT9"

];

base = toUpper(base);

foreach(rsvd; reservedNames)

if(base == rsvd)

return false;

return true;

}

//---------------------------------------------------------------------------

// Class: CVsModalState

// Manage Modal State

//---------------------------------------------------------------------------

class CVsModalState

{

public:

this(bool bDisableDlgOwnerHwnd = false)

{

m\_hwnd = null;

m\_bDisabledHwnd = false;

// Need to get dialog owner hwnd prior to enabling modeless false

auto srpUIManager = queryService!(IVsUIShell);

if(srpUIManager)

{

srpUIManager.GetDialogOwnerHwnd(&m\_hwnd);

srpUIManager.Release();

}

if(m\_hwnd == null)

{

//assert(false);

m\_hwnd = GetActiveWindow();

}

EnableModeless(false);

if(bDisableDlgOwnerHwnd && IsWindowEnabled(m\_hwnd))

{

EnableWindow(m\_hwnd, FALSE);

m\_bDisabledHwnd = true;

}

}

~this()

{

if(m\_bDisabledHwnd)

EnableWindow(m\_hwnd, TRUE);

EnableModeless(TRUE);

}

HWND GetDialogOwnerHwnd()

{

return m\_hwnd;

}

protected:

HRESULT EnableModeless(bool fEnable)

{

HRESULT hr = S\_OK;

auto srpUIManager = queryService!(IVsUIShell);

if(srpUIManager)

{

hr = srpUIManager.EnableModeless(fEnable);

srpUIManager.Release();

}

return hr;

}

HWND m\_hwnd; // owner window

bool m\_bDisabledHwnd; // TRUE if we disabled m\_hwnd;

}

int UtilMessageBox(string text, uint nType, string caption)

{

auto wtext = toUTF16z(text);

auto wcaption = toUTF16z(caption);

scope CVsModalState modalstate = new CVsModalState;

return MessageBoxW(modalstate.GetDialogOwnerHwnd(), wtext, wcaption, nType);

}

struct DROPFILES

{

DWORD pFiles; // offset of file list

POINT pt; // drop point (coordinates depend on fNC)

BOOL fNC; // see below

BOOL fWide; // TRUE if file contains wide characters, FALSE otherwise

}

//-----------------------------------------------------------------------------

// Returns a cstring array populated with the files from a PROJREF drop. Note that

// we can't use the systems DragQueryFile() functions because they will NOT work

// on win9x with unicode strings. Returns the count of files. The format looks like

// the following: DROPFILES structure with pFiles member containing the offset to

// the list of files:

// ----------------------------------------------------------------------------

// |{DROPFILES structure}|ProjRefItem1|0|ProjRefItem2|0|.......|ProjRefItemN|0|0|

// ----------------------------------------------------------------------------

//-----------------------------------------------------------------------------

int UtilGetFilesFromPROJITEMDrop(HGLOBAL h, ref string[] rgFiles)

{

LPVOID pv = .GlobalLock(h);

if (!pv)

return 0;

DROPFILES\* pszDropFiles = cast(DROPFILES\*)pv;

// It better be marked unicode

assert(pszDropFiles.fWide);

if (pszDropFiles.fWide)

{

// The first member of the structure contains the offset to the files

wchar\* wzBuffer = cast(wchar\*)(cast(byte\*)pszDropFiles + pszDropFiles.pFiles);

// We go until \*wzBuffer is null since we don't allow empty strings.

while(\*wzBuffer)

{

int len = wcslen(wzBuffer);

assert(len);

string file = toUTF8(wzBuffer[0..len]);

rgFiles ~= file;

wzBuffer += len + 1;

}

}

.GlobalUnlock(h);

return rgFiles.length;

}

wstring UtilGetStringFromHGLOBAL(HGLOBAL h)

{

LPVOID pv = .GlobalLock(h);

if (!pv)

return "";

wstring ws = to\_wstring(cast(wchar\*) pv);

.GlobalUnlock(h);

return ws;

}

//----------------------------------------------------------------------------

// Returns TRUE if Shell is in command line (non-interactive) mode

//----------------------------------------------------------------------------

bool UtilShellInCmdLineMode()

{

auto pIVsShell = ComPtr!(IVsShell)(queryService!(IVsShell), false);

if(pIVsShell)

{

VARIANT var;

if(SUCCEEDED(pIVsShell.GetProperty(VSSPROPID\_IsInCommandLineMode, &var)))

return var.boolVal != 0;

}

return false;

}

//-----------------------------------------------------------------------------

// Displays the last error set in the shell

//-----------------------------------------------------------------------------

void UtilReportErrorInfo(HRESULT hr)

{

// Filter out bogus hr's where we shouldn't be displaying an error.

if(hr != OLE\_E\_PROMPTSAVECANCELLED)

{

BOOL fInExt = FALSE;

if(dte.IVsExtensibility ext = queryService!(dte.IVsExtensibility))

{

scope(exit) release(ext);

ext.IsInAutomationFunction(&fInExt);

if(fInExt || UtilShellInCmdLineMode())

return;

auto pIVsUIShell = ComPtr!(IVsUIShell)(queryService!(IVsUIShell), false);

if(pIVsUIShell)

pIVsUIShell.ReportErrorInfo(hr);

}

}

}

int ShowContextMenu(UINT iCntxtMenuID, in GUID\* GroupGuid, IOleCommandTarget pIOleCmdTarg)

{

auto srpUIManager = queryService!(IVsUIShell);

if(!srpUIManager)

return E\_FAIL;

scope(exit) release(srpUIManager);

POINT pnt;

GetCursorPos(&pnt);

POINTS pnts = POINTS(cast(short)pnt.x, cast(short)pnt.y);

int hr = srpUIManager.ShowContextMenu(0, GroupGuid, iCntxtMenuID, &pnts, pIOleCmdTarg);

return hr;

}

//-----------------------------------------------------------------------------

CHierNode searchNode(CHierNode root, bool delegate(CHierNode) pred, bool fDisplayOnly = true)

{

if(!root)

return null;

if(pred(root))

return root;

for(CHierNode node = root.GetHeadEx(fDisplayOnly); node; node = node.GetNext(fDisplayOnly))

if(CHierNode n = searchNode(node, pred, fDisplayOnly))

return n;

return null;

}

///////////////////////////////////////////////////////////////////////

@property I queryService(SVC,I)()

{

if(!visuald.dpackage.Package.s\_instance)

return null;

IServiceProvider sp = visuald.dpackage.Package.s\_instance.getServiceProvider();

if(!sp)

return null;

I svc;

if(FAILED(sp.QueryService(&SVC.iid, &I.iid, cast(void \*\*)&svc)))

return null;

return svc;

}

@property I queryService(I)()

{

return queryService!(I,I);

}

///////////////////////////////////////////////////////////////////////////////

// VsLocalCreateInstance

///////////////////////////////////////////////////////////////////////////////

I VsLocalCreateInstance(I)(const GUID\* clsid, DWORD dwFlags)

{

if(ILocalRegistry srpLocalReg = queryService!ILocalRegistry())

{

scope(exit) release(srpLocalReg);

IUnknown punkOuter = null;

I inst;

if(FAILED(srpLocalReg.CreateInstance(\*clsid, punkOuter, &I.iid, dwFlags,

cast(void\*\*) &inst)))

return null;

return inst;

}

return null;

}

///////////////////////////////////////////////////////////////////////////////

dte2.DTE2 GetDTE()

{

dte.\_DTE \_dte = queryService!(dte.\_DTE);

if(!\_dte)

return null;

scope(exit) release(\_dte);

dte2.DTE2 spvsDTE = qi\_cast!(dte2.DTE2)(\_dte);

return spvsDTE;

}

int GetDTE(dte.DTE \*lppaReturn)

{

dte.\_DTE \_dte = queryService!(dte.\_DTE);

if(!\_dte)

return returnError(E\_NOINTERFACE);

scope(exit) \_dte.Release();

return \_dte.get\_DTE(lppaReturn);

}

string getStringProperty(dte.Properties props, string propName, string def = null)

{

VARIANT index;

dte.Property prop;

index.vt = VT\_BSTR;

index.bstrVal = allocBSTR(propName);

HRESULT hr = props.Item(index, &prop);

detachBSTR(index.bstrVal);

if(FAILED(hr) || !prop)

return def;

scope(exit) release(prop);

VARIANT var;

hr = prop.get\_Value(&var);

if(var.vt != VT\_BSTR)

return def;

if(FAILED(hr))

return def;

return detachBSTR(var.bstrVal);

}

int getIntProperty(dte.Properties props, string propName, int def = -1)

{

VARIANT index;

dte.Property prop;

index.vt = VT\_BSTR;

index.bstrVal = allocBSTR(propName);

HRESULT hr = props.Item(index, &prop);

detachBSTR(index.bstrVal);

if(FAILED(hr) || !prop)

return def;

scope(exit) release(prop);

VARIANT var;

hr = prop.get\_Value(&var);

if(FAILED(hr))

return def;

if(var.vt == VT\_I2 || var.vt == VT\_UI2)

return var.iVal;

if(var.vt == VT\_INT || var.vt == VT\_I4 || var.vt == VT\_UI4 || var.vt == VT\_UINT)

return var.intVal;

return def;

}

string getEnvironmentFont(out int fontSize, out int charSet)

{

dte.\_DTE \_dte = queryService!(dte.\_DTE);

if(!\_dte)

return null;

scope(exit) release(\_dte);

dte.Properties props;

BSTR bprop = allocBSTR("FontsAndColors");

BSTR bpage = allocBSTR("Dialogs and Tool Windows");

HRESULT hr = \_dte.get\_Properties(bprop, bpage, &props);

detachBSTR(bprop);

detachBSTR(bpage);

if(FAILED(hr) || !props)

return null;

scope(exit) release(props);

string family = getStringProperty(props, "FontFamily");

fontSize = getIntProperty(props, "FontSize", 10);

charSet = getIntProperty(props, "FontCharacterSet", 1);

/+

IDispatch obj;

hr = prop.Object(&obj);

if(FAILED(hr) || !obj)

return null;

scope(exit) release(obj);

dte.FontsAndColorsItems faci = qi\_cast!(dte.FontsAndColorsItems)(obj);

if(!faci)

return null;

scope(exit) release(faci);

dte.ColorableItems ci;

index.bstrVal = allocBSTR("Plain Text");

hr = faci.Item(index, &ci);

detachBSTR(index.bstrVal);

if(FAILED(hr) || !ci)

return null;

BSTR wname;

ci.Name(&wname);

string name = detachBSTR(wname);

dte.\_FontsAndColors fac = qi\_cast!(dte.\_FontsAndColors)(ci);

fac = release(fac);

fac = qi\_cast!(dte.\_FontsAndColors)(faci);

fac = release(fac);

+/

return family;

}

void updateEnvironmentFont()

{

IUIHostLocale locale = queryService!(IUIHostLocale);

if(locale)

{

scope(exit) release(locale);

if(SUCCEEDED(locale.GetDialogFont(&dialogLogFont)))

return;

}

int size;

int charset;

string font = getEnvironmentFont(size, charset);

if(font.length)

{

HDC hDDC = GetDC(GetDesktopWindow());

int nHeight = -MulDiv(size, GetDeviceCaps(hDDC, LOGPIXELSY), 72);

dialogLogFont.lfHeight = nHeight;

dialogLogFont.lfCharSet = cast(ubyte)charset;

dialogLogFont.lfFaceName[] = to!wstring(font)[];

}

}

////////////////////////////////////////////////////////////////////////

IVsTextLines GetCurrentTextBuffer(IVsTextView\* pview)

{

IVsTextManager textmgr = queryService!(VsTextManager, IVsTextManager);

if(!textmgr)

return null;

scope(exit) release(textmgr);

IVsTextView view;

if(textmgr.GetActiveView(false, null, &view) != S\_OK)

return null;

scope(exit) release(view);

if(pview)

\*pview = addref(view);

IVsTextLines buffer;

view.GetBuffer(&buffer);

return buffer;

}

////////////////////////////////////////////////////////////////////////

string GetSolutionFilename()

{

IVsSolution srpSolution = queryService!(IVsSolution);

if(srpSolution)

{

scope(exit) srpSolution.Release();

BSTR pbstrSolutionFile;

if(srpSolution.GetSolutionInfo(null, &pbstrSolutionFile, null) == S\_OK)

return detachBSTR(pbstrSolutionFile);

}

return "";

}

////////////////////////////////////////////////////////////////////////

HRESULT FindFileInSolution(IVsUIShellOpenDocument pIVsUIShellOpenDocument, string filename, string srcfile,

out BSTR bstrAbsPath)

{

auto wstrPath = \_toUTF16z(filename);

HRESULT hr;

hr = pIVsUIShellOpenDocument.SearchProjectsForRelativePath(RPS\_UseAllSearchStrategies, wstrPath, &bstrAbsPath);

if(hr != S\_OK || !bstrAbsPath || !isAbsolute(to\_string(bstrAbsPath)))

{

// search import paths

string[] imps = GetImportPaths(srcfile);

foreach(imp; imps)

{

string file = makeFilenameCanonical(filename, imp);

if(std.file.exists(file))

{

detachBSTR(bstrAbsPath);

bstrAbsPath = allocBSTR(file);

hr = S\_OK;

break;

}

}

}

return hr;

}

HRESULT FindFileInSolution(string filename, string srcfile, out string absPath)

{

// Get the IVsUIShellOpenDocument service so we can ask it to open a doc window

IVsUIShellOpenDocument pIVsUIShellOpenDocument = queryService!(IVsUIShellOpenDocument);

if(!pIVsUIShellOpenDocument)

return returnError(E\_FAIL);

scope(exit) release(pIVsUIShellOpenDocument);

BSTR bstrAbsPath;

HRESULT hr = FindFileInSolution(pIVsUIShellOpenDocument, filename, srcfile, bstrAbsPath);

if(hr != S\_OK)

return returnError(hr);

absPath = detachBSTR(bstrAbsPath);

return S\_OK;

}

HRESULT OpenFileInSolution(string filename, int line, int col = 0, string srcfile = "", bool adjustLineToChanges = false)

{

// Get the IVsUIShellOpenDocument service so we can ask it to open a doc window

IVsUIShellOpenDocument pIVsUIShellOpenDocument = queryService!(IVsUIShellOpenDocument);

if(!pIVsUIShellOpenDocument)

return returnError(E\_FAIL);

scope(exit) release(pIVsUIShellOpenDocument);

BSTR bstrAbsPath;

HRESULT hr = FindFileInSolution(pIVsUIShellOpenDocument, filename, srcfile, bstrAbsPath);

if(hr != S\_OK)

return returnError(hr);

scope(exit) detachBSTR(bstrAbsPath);

IVsWindowFrame srpIVsWindowFrame;

hr = pIVsUIShellOpenDocument.OpenDocumentViaProject(bstrAbsPath, &LOGVIEWID\_Primary, null, null, null,

&srpIVsWindowFrame);

if(FAILED(hr))

hr = pIVsUIShellOpenDocument.OpenStandardEditor(

/\* [in] VSOSEFLAGS grfOpenStandard \*/ OSE\_ChooseBestStdEditor,

/\* [in] LPCOLESTR pszMkDocument \*/ bstrAbsPath,

/\* [in] REFGUID rguidLogicalView \*/ &LOGVIEWID\_Primary,

/\* [in] LPCOLESTR pszOwnerCaption \*/ \_toUTF16z("%3"),

/\* [in] IVsUIHierarchy \*pHier \*/ null,

/\* [in] VSITEMID itemid \*/ 0,

/\* [in] IUnknown \*punkDocDataExisting \*/ DOCDATAEXISTING\_UNKNOWN,

/\* [in] IServiceProvider \*pSP \*/ null,

/\* [out, retval] IVsWindowFrame \*\*ppWindowFrame \*/ &srpIVsWindowFrame);

if(FAILED(hr) || !srpIVsWindowFrame)

return returnError(hr);

scope(exit) release(srpIVsWindowFrame);

srpIVsWindowFrame.Show();

VARIANT var;

hr = srpIVsWindowFrame.GetProperty(VSFPROPID\_DocData, &var);

if(FAILED(hr) || var.vt != VT\_UNKNOWN || !var.punkVal)

return returnError(E\_FAIL);

scope(exit) release(var.punkVal);

IVsTextLines textBuffer = qi\_cast!IVsTextLines(var.punkVal);

if(!textBuffer)

if(auto bufferProvider = qi\_cast!IVsTextBufferProvider(var.punkVal))

{

bufferProvider.GetTextBuffer(&textBuffer);

release(bufferProvider);

}

if(!textBuffer)

return returnError(E\_FAIL);

scope(exit) release(textBuffer);

if(line < 0)

return S\_OK;

if(adjustLineToChanges)

if(auto src = Package.GetLanguageService().GetSource(textBuffer))

line = src.adjustLineNumberSinceLastBuild(line, false);

return NavigateTo(textBuffer, line, col, line, col);

}

HRESULT NavigateTo(IVsTextBuffer textBuffer, int line1, int col1, int line2, int col2)

{

IVsTextManager textmgr = queryService!(VsTextManager, IVsTextManager);

if(!textmgr)

return returnError(E\_FAIL);

scope(exit) release(textmgr);

return textmgr.NavigateToLineAndColumn(textBuffer, &LOGVIEWID\_Primary, line1, col1, line2, col2);

}

HRESULT OpenFileInSolutionWithScope(string fname, int line, int col, string scop, bool adjustLineToChanges = false)

{

HRESULT hr = OpenFileInSolution(fname, line, col, "", adjustLineToChanges);

if(hr != S\_OK && !isAbsolute(fname) && scop.length)

{

// guess import path from filename (e.g. "src\core\mem.d") and

// scope (e.g. "core.mem.gc.Proxy") to try opening

// the file ("core\mem.d")

string inScope = toLower(scop);

string path = normalizeDir(dirName(toLower(fname)));

inScope = replace(inScope, ".", "\\");

int i;

for(i = 1; i < path.length; i++)

if(startsWith(inScope, path[i .. $]))

break;

if(i < path.length)

{

fname = fname[i .. $];

hr = OpenFileInSolution(fname, line, col, "", adjustLineToChanges);

}

}

return hr;

}

////////////////////////////////////////////////////////////////////////

string commonProjectFolder(Project proj)

{

string workdir = normalizeDir(dirName(proj.GetFilename()));

string path = workdir;

searchNode(proj.GetRootNode(), delegate (CHierNode n)

{

if(CFileNode file = cast(CFileNode) n)

path = commonParentDir(path, makeFilenameAbsolute(file.GetFilename(), workdir));

return false;

});

return path;

}

////////////////////////////////////////////////////////////////////////

string copyProjectFolder(Project proj, string ncommonpath)

{

string path = commonProjectFolder(proj);

if (path.length == 0)

return null;

string npath = normalizeDir(ncommonpath);

string workdir = normalizeDir(dirName(proj.GetFilename()));

searchNode(proj.GetRootNode(), delegate (CHierNode n)

{

if(CFileNode file = cast(CFileNode) n)

{

string fname = makeFilenameAbsolute(file.GetFilename(), workdir);

string nname = npath ~ fname[path.length .. $];

mkdirRecurse(dirName(nname));

copy(fname, nname);

}

return false;

});

return npath;

}

////////////////////////////////////////////////////////////////////////

string GetFolderPath(CFolderNode folder)

{

string path;

while(folder && !cast(CProjectNode) folder)

{

path = "\\" ~ folder.GetName() ~ path;

folder = cast(CFolderNode) folder.GetParent();

}

return path;

}

///////////////////////////////////////////////////////////////

// returns addref'd Config

Config getProjectConfig(string file)

{

if(file.length == 0)

return null;

auto srpSolution = queryService!(IVsSolution);

scope(exit) release(srpSolution);

auto solutionBuildManager = queryService!(IVsSolutionBuildManager)();

scope(exit) release(solutionBuildManager);

if(srpSolution && solutionBuildManager)

{

bool isJSON = toLower(extension(file)) == ".json";

auto wfile = \_toUTF16z(file);

IEnumHierarchies pEnum;

if(srpSolution.GetProjectEnum(EPF\_LOADEDINSOLUTION|EPF\_MATCHTYPE, &g\_projectFactoryCLSID, &pEnum) == S\_OK)

{

scope(exit) release(pEnum);

IVsHierarchy pHierarchy;

while(pEnum.Next(1, &pHierarchy, null) == S\_OK)

{

scope(exit) release(pHierarchy);

IVsProjectCfg activeCfg;

scope(exit) release(activeCfg);

if(isJSON)

{

if(solutionBuildManager.FindActiveProjectCfg(null, null, pHierarchy, &activeCfg) == S\_OK)

{

if(Config cfg = qi\_cast!Config(activeCfg))

{

string[] files;

if(cfg.addJSONFiles(files))

foreach(f; files)

if(CompareFilenames(f, file) == 0)

return cfg;

release(cfg);

}

}

}

else

{

VSITEMID itemid;

if(pHierarchy.ParseCanonicalName(wfile, &itemid) == S\_OK)

{

if(solutionBuildManager.FindActiveProjectCfg(null, null, pHierarchy, &activeCfg) == S\_OK)

{

if(Config cfg = qi\_cast!Config(activeCfg))

return cfg;

}

}

}

}

}

}

return null;

}

Config getCurrentStartupConfig()

{

auto solutionBuildManager = queryService!(IVsSolutionBuildManager)();

scope(exit) release(solutionBuildManager);

if(solutionBuildManager)

{

IVsHierarchy pHierarchy;

if(solutionBuildManager.get\_StartupProject(&pHierarchy) == S\_OK)

{

scope(exit) release(pHierarchy);

IVsProjectCfg activeCfg;

if(solutionBuildManager.FindActiveProjectCfg(null, null, pHierarchy, &activeCfg) == S\_OK)

{

scope(exit) release(activeCfg);

if(Config cfg = qi\_cast!Config(activeCfg))

return cfg;

}

}

}

return null;

}

////////////////////////////////////////////////////////////////////////

string[] GetImportPaths(string file)

{

string[] imports;

if(Config cfg = getProjectConfig(file))

{

scope(exit) release(cfg);

ProjectOptions opt = cfg.GetProjectOptions();

string projectpath = cfg.GetProjectDir();

string imp = opt.imppath;

imp = opt.replaceEnvironment(imp, cfg);

imports = tokenizeArgs(imp);

string addopts = opt.replaceEnvironment(opt.additionalOptions, cfg);

addunique(imports, GlobalOptions.getOptionImportPaths(addopts, projectpath));

foreach(ref i; imports)

i = makeDirnameCanonical(unquoteArgument(i), projectpath);

addunique(imports, projectpath);

}

imports ~= Package.GetGlobalOptions().getImportPaths();

return imports;

}

////////////////////////////////////////////////////////////////////////

const(wchar)\* \_toFilter(string filter)

{

wchar\* s = \_toUTF16z(filter);

for(wchar\*p = s; \*p; p++)

if(\*p == '|')

\*p = 0;

return s;

}

string getOpenFileDialog(HWND hwnd, string title, string dir, string filter)

{

string file;

auto pIVsUIShell = ComPtr!(IVsUIShell)(queryService!(IVsUIShell));

if(pIVsUIShell)

{

wchar[260] filename;

VSOPENFILENAMEW ofn;

ofn.lStructSize = ofn.sizeof;

ofn.hwndOwner = hwnd;

ofn.pwzDlgTitle = \_toUTF16z(title);

ofn.pwzFileName = filename.ptr;

ofn.nMaxFileName = 260;

ofn.pwzInitialDir = \_toUTF16z(dir);

ofn.pwzFilter = \_toFilter(filter);

HRESULT hr = pIVsUIShell.GetOpenFileNameViaDlg(&ofn);

if(hr != S\_OK)

return "";

file = to!string(filename);

}

return file;

}

string getSaveFileDialog(HWND hwnd, string title, string dir, string filter)

{

string file;

auto pIVsUIShell = ComPtr!(IVsUIShell)(queryService!(IVsUIShell));

if(pIVsUIShell)

{

wchar[260] filename;

VSSAVEFILENAMEW ofn;

ofn.lStructSize = ofn.sizeof;

ofn.hwndOwner = hwnd;

ofn.pwzDlgTitle = \_toUTF16z(title);

ofn.pwzFileName = filename.ptr;

ofn.nMaxFileName = 260;

ofn.pwzInitialDir = \_toUTF16z(dir);

ofn.pwzFilter = \_toFilter(filter);

HRESULT hr = pIVsUIShell.GetSaveFileNameViaDlg(&ofn);

if(hr != S\_OK)

return "";

file = to!string(filename);

}

return file;

}

// This file is part of Visual D

//

// Visual D integrates the D programming language into Visual Studio

// Copyright (c) 2010 by Rainer Schuetze, All Rights Reserved

//

// Distributed under the Boost Software License, Version 1.0.

// See accompanying file LICENSE\_1\_0.txt or copy at <http://www.boost.org/LICENSE_1_0.txt>

module visuald.intellisense;

import std.json;

import std.file;

import std.utf;

import std.datetime;

import std.conv;

import std.string;

import std.algorithm;

import std.regex;

import std.array;

import std.path : baseName, stripExtension;

//import stdext.fred;

import stdext.path;

import stdext.array;

import core.memory;

import core.demangle;

import visuald.windows;

import sdk.port.vsi;

import sdk.vsi.vsshell;

import visuald.dpackage;

import visuald.dlangsvc;

import visuald.config;

import visuald.comutil;

import visuald.logutil;

import visuald.hierutil;

import visuald.fileutil;

import visuald.pkgutil;

import visuald.stringutil;

import vdc.lexer;

enum MatchType

{

Exact,

CaseInsensitive,

StartsWith,

RegExp

}

struct SearchData

{

string[] names;

Regex!char[] res;

enum

{

kFieldName = 1 << 0,

kFieldType = 1 << 1,

kFieldScope = 1 << 2,

kFieldDeco = 1 << 3,

}

ubyte searchFields = kFieldName;

bool wholeWord;

bool caseSensitive;

bool useRegExp;

bool noDupsOnSameLine;

bool findQualifiedName;

bool init(string[] nms)

{

try

{

if(useRegExp)

foreach(string nm; nms)

res ~= regex(nm, caseSensitive ? "" : "i");

else

names = nms;

}

catch(Exception)

{

return false;

}

return true;

}

string getQualifiedName(JSONscope \*sc, JSONValue[string] obj)

{

string name;

if(JSONValue\* n = "name" in obj)

if(n.type == JSON\_TYPE.STRING)

name = n.str;

string scname = sc.toString();

if(JSONValue\* n = "kind" in obj)

if(n.type == JSON\_TYPE.STRING)

if(n.str == "module")

name = "";

if(name.length == 0)

name = scname;

else if (scname.length != 0)

name = scname ~ "." ~ name;

return name;

}

bool matchDefinition(JSONscope \*sc, JSONValue[string] obj)

{

if(findQualifiedName && names.length > 0)

return sc.toString() == names[0];

if((!useRegExp && names.length == 0) || (useRegExp && res.length == 0))

return true;

string name, type, deco, inScope;

if(searchFields & kFieldName)

if(JSONValue\* n = "name" in obj)

if(n.type == JSON\_TYPE.STRING)

name = caseSensitive ? n.str : toLower(n.str);

if(searchFields & kFieldType)

if(JSONValue\* typ = "type" in obj)

if(typ.type == JSON\_TYPE.STRING)

type = caseSensitive ? typ.str : toLower(typ.str);

if(searchFields & kFieldDeco)

if(JSONValue\* dec = "deco" in obj)

if(dec.type == JSON\_TYPE.STRING)

deco = caseSensitive ? dec.str : toLower(dec.str);

if(searchFields & kFieldScope)

inScope = sc ? (caseSensitive ? sc.toString() : toLower(sc.toString())) : "";

return matchNames(name, type, deco, inScope);

}

bool matchDefinition(BrowseNode node)

{

if(findQualifiedName && names.length > 0)

return node.GetScope() == names[0];

if((!useRegExp && names.length == 0) || (useRegExp && res.length == 0))

return true;

string name, type, deco, inScope;

if(searchFields & kFieldName)

name = caseSensitive ? node.name : toLower(node.name);

if(searchFields & kFieldType)

type = caseSensitive ? node.\_type : toLower(node.\_type);

if(searchFields & kFieldDeco)

deco = caseSensitive ? node.deco : toLower(node.deco);

if(searchFields & kFieldScope)

inScope = caseSensitive ? node.GetScope() : toLower(node.GetScope());

return matchNames(name, type, deco, inScope);

}

void addDefinition(ref Definition[] defs, ref Definition def)

{

bool add = true;

if(noDupsOnSameLine)

{

foreach(d; defs)

if(d.filename == def.filename && d.line == def.line)

add = false;

}

if(add)

defs ~= def;

}

bool pruneSubtree(JSONscope \*sc, JSONValue[string] obj)

{

if(findQualifiedName && names.length > 0)

{

string name = sc.toString();

return !startsWith(names[0], name);

}

return false;

}

bool pruneSubtree(BrowseNode node)

{

if(findQualifiedName && names.length > 0)

{

string name = node.GetScope();

return !startsWith(names[0], name);

}

return false;

}

static bool isIdentChar(dchar ch)

{

return dLex.isIdentifierCharOrDigit(ch);

}

static bool isWordBoundary(dchar ch1, dchar ch2)

{

return !isIdentChar(ch1) || !isIdentChar(ch2);

}

bool matchNames(string name, string type, string deco, string inScope)

{

bool matches = false;

if(useRegExp)

{

bool matchRegex(string txt, Regex!char re)

{

auto m = match(name, re);

if(m.empty() || m.hit.length == 0)

return false;

if(!wholeWord)

return true;

foreach(mx; m)

if((mx.pre.length == 0 || isWordBoundary(mx.pre[$-1], mx.hit[0])) &&

(mx.post.length == 0 || isWordBoundary(mx.post[0], mx.hit[$-1])))

return true;

return false;

}

for(int i = 0; i < res.length; i++)

{

if(searchFields & kFieldName)

if(matchRegex(name, res[i]))

continue;

if(searchFields & kFieldType)

if(matchRegex(type, res[i]))

continue;

if(searchFields & kFieldDeco)

if(matchRegex(deco, res[i]))

continue;

if(searchFields & kFieldScope)

if(matchRegex(inScope, res[i]))

continue;

return false;

}

}

else

{

bool matchString(string txt, string str)

{

CaseSensitive cs = caseSensitive ? CaseSensitive.yes : CaseSensitive.no;

int pos = 0;

int p = pos + indexOfPath(name[pos..$], str, cs);

while(p >= pos)

{

if(!wholeWord)

return true;

if((p == 0 || isWordBoundary(txt[p-1], txt[0])) &&

(p + str.length >= txt.length || isWordBoundary(txt[p + str.length - 1], txt[p + str.length])))

return true;

pos = p + 1;

p = pos + indexOfPath(name[pos..$], str, cs);

}

return false;

}

for(int i = 0; i < names.length; i++)

{

if(searchFields & kFieldName)

if(matchString(name, names[i]))

continue;

if(searchFields & kFieldType)

if(matchString(type, names[i]))

continue;

if(searchFields & kFieldDeco)

if(matchString(deco, names[i]))

continue;

if(searchFields & kFieldScope)

if(matchString(inScope, names[i]))

continue;

return false;

}

}

return true;

}

}

struct JSONscope

{

JSONscope\* parent;

string name;

string toString()

{

string nm = name;

if(parent && nm.length > 0)

nm = parent.toString() ~ "." ~ nm;

else if(parent)

nm = parent.toString();

return nm;

}

}

// filter out stuff written by dmd 2.062alpha

bool isDeclarationKind(string kind)

{

switch(kind)

{

case "import":

case "static import":

case "alias this":

case "static assert":

case "template instance":

case "mixin":

return false;

default:

return true;

}

}

string demangleType(string type, string name)

{

string sym = "\_D7\_\_Sym\_\_" ~ type;

string s = cast(string) demangle(sym);

if(s == sym) // cannot demangle

return type;

s = s.replace("\_\_Sym\_\_", "");

return s;

}

void getDeclarationInfo(D)(D def, JSONValue[string] obj)

{

if(JSONValue\* n = "name" in obj)

if(n.type == JSON\_TYPE.STRING)

def.name = n.str;

if(JSONValue\* ln = "line" in obj)

if(ln.type == JSON\_TYPE.INTEGER)

def.line = cast(int)ln.integer - 1;

if(JSONValue\* typ = "type" in obj)

{

if(typ.type == JSON\_TYPE.STRING)

def.\_type = typ.str;

}

// dmd 2.062:

if(JSONValue\* dec = "deco" in obj)

if(dec.type == JSON\_TYPE.STRING)

def.deco = dec.str;

}

class LibraryInfo

{

bool readJSON(string fileName)

{

try

{

string text = cast(string) std.file.read(fileName);

size\_t decidx = 0;

if(decode(text, decidx) == 0xfeff)

text = text[decidx..$];

mModules = parseJSON(text);

mFilename = fileName;

mModified = timeLastModified(fileName);

return true;

}

catch(JSONException rc)

{

string msg = rc.toString();

writeToBuildOutputPane(fileName ~ ": " ~ msg);

logCall("EXCEPTION: " ~ msg);

}

catch(UTFException rc)

{

string msg = rc.toString();

writeToBuildOutputPane(fileName ~ ": " ~ msg);

logCall("EXCEPTION: " ~ msg);

}

catch(FileException rc)

{

string msg = rc.toString();

writeToBuildOutputPane(fileName ~ ": " ~ msg);

logCall("EXCEPTION: " ~ msg);

}

return false;

}

// dg\_match returns:

// 0 - continue search

// 1 - stop search

// 2 - continue search, but prune subtree

bool iterateObjects(int delegate(string filename, JSONscope\* sc, JSONValue[string] object) dg\_match)

{

if(mModules.type == JSON\_TYPE.ARRAY)

{

JSONValue[] modules = mModules.array;

foreach(JSONValue mod; modules)

{

if(mod.type == JSON\_TYPE.OBJECT)

{

string filename;

string modname;

JSONValue[string] object = mod.object;

if(JSONValue\* v = "file" in object)

if(v.type == JSON\_TYPE.STRING)

filename = v.str;

if(JSONValue\* v = "name" in object)

if(v.type == JSON\_TYPE.STRING)

modname = v.str;

int iterate(JSONValue[string] object, JSONscope\* sc)

{

int res = dg\_match(filename, sc, object);

if(res == 1)

return 1;

if(res == 2)

return 0;

if(JSONValue\* m = "members" in object)

if(m.type == JSON\_TYPE.ARRAY)

{

JSONValue[] members = m.array;

foreach(member; members)

{

if(member.type == JSON\_TYPE.OBJECT)

{

string nm;

JSONValue[string] memberobj = member.object;

if(JSONValue\* n = "name" in memberobj)

if(n.type == JSON\_TYPE.STRING)

nm = n.str;

JSONscope msc = JSONscope(sc, nm);

res = iterate(memberobj, &msc);

if(res > 0)

return res;

}

}

}

return 0;

}

JSONscope sc = JSONscope(null, modname);

if(iterate(object, &sc))

return true;

}

}

}

return false;

}

JSONValue[] getModules()

{

if(mModules.type == JSON\_TYPE.ARRAY)

return mModules.array;

return null;

}

Definition[] findDefinition(ref SearchData sd)

{

Definition[] defs;

//GC.disable();

debug(FINDDEF) {

int cnt = 0;

int cntKind = 0;

int cntLine = 0;

int cntType = 0;

int countDef(string filename, JSONscope\* sc, JSONValue[string] memberobj)

{

if(sd.pruneSubtree(sc, memberobj))

return 2;

if(sd.matchDefinition(sc, memberobj))

{

if(JSONValue\* n = "name" in memberobj)

if(n.type == JSON\_TYPE.STRING)

cnt++;

if(JSONValue\* k = "kind" in memberobj)

if(k.type == JSON\_TYPE.STRING)

cntKind++;

if(JSONValue\* ln = "line" in memberobj)

if(ln.type == JSON\_TYPE.INTEGER)

cntLine++;

if(JSONValue\* typ = "type" in memberobj)

if(typ.type == JSON\_TYPE.STRING)

cntType++;

}

return 0;

}

iterateObjects(&countDef);

}

int findDef(string filename, JSONscope\* sc, JSONValue[string] memberobj)

{

if(sd.pruneSubtree(sc, memberobj))

return 2;

if(sd.matchDefinition(sc, memberobj))

{

Definition def;

def.filename = filename;

def.inScope = sc ? sc.toString() : "";

if(JSONValue\* k = "kind" in memberobj)

if(k.type == JSON\_TYPE.STRING)

def.kind = k.str;

if(!isDeclarationKind(def.kind))

return 2;

getDeclarationInfo(def, memberobj);

sd.addDefinition(defs, def);

}

return 0;

}

iterateObjects(&findDef);

//GC.enable();

return defs;

}

string[] findCompletions(ref SearchData sd)

{

string[] cplts;

int findCplt(string filename, JSONscope\* sc, JSONValue[string] memberobj)

{

if(JSONValue\* n = "name" in memberobj)

if(n.type == JSON\_TYPE.STRING)

if(startsWith(n.str, sd.names[0]))

{

// strip template arguments and constraint

string s = n.str;

int pos = indexOf(s, '(');

if(pos >= 0)

s = s[0..pos];

addunique(cplts, s);

}

return 0;

}

iterateObjects(&findCplt);

return cplts;

}

JSONValue mModules;

string mFilename;

SysTime mModified;

}

struct ParameterInfo

{

string rettype;

string[] name;

string[] display;

string[] desc;

bool initialize(string type)

{

wstring text = to!wstring(type);

TokenInfo[] lineInfo = dLex.ScanLine(Lexer.State.kWhite, text);

if(lineInfo.length == 0)

return false;

int pos = lineInfo.length - 1;

if(text[lineInfo[pos].StartIndex .. lineInfo[pos].EndIndex] != ")")

return false; // not a function

int braceLevel = 1;

pos--;

string ident;

int endpos = lineInfo[pos].EndIndex;

void prependParam()

{

wstring wdisp = text[lineInfo[pos].EndIndex .. endpos];

string disp = strip(to!string(wdisp));

if(disp.length)

{

name = ident ~ name;

display = disp ~ display;

desc = "" ~ desc;

ident = "";

}

endpos = lineInfo[pos].StartIndex;

}

while(pos > 0 && braceLevel > 0)

{

wstring tok = text[lineInfo[pos].StartIndex .. lineInfo[pos].EndIndex];

if(ident.length == 0 && lineInfo[pos].type == TokenCat.Identifier)

ident = to!string(tok);

else if (tok == ",")

prependParam();

else if(tok == ")")

braceLevel++;

else if(tok == "(")

{

braceLevel--;

if(braceLevel == 0)

prependParam();

}

pos--;

}

wstring wret = text[0 .. endpos];

rettype = strip(to!string(wret));

return braceLevel == 0;

}

}

struct Definition

{

string name;

string kind;

string filename;

string deco;

string help;

int line;

private string \_type;

@property string type() const

{

if(\_type.length == 0 && deco.length)

(cast()this).\_type = demangleType(deco, name);

return \_type;

}

void setType(string t)

{

\_type = t;

}

string inScope; // enclosing scope

ParameterInfo\* paramInfo;

ParameterInfo\* GetParamInfo()

{

if(!paramInfo)

{

paramInfo = new ParameterInfo;

paramInfo.initialize(type);

}

return paramInfo;

}

string GetReturnType()

{

return GetParamInfo().rettype;

}

int GetParameterCount()

{

return GetParamInfo().name.length;

}

void GetParameterInfo(int parameter, out string name, out string display, out string description)

{

ParameterInfo\* info = GetParamInfo();

if(parameter < 0 || parameter >= info.name.length)

return;

name = info.name[parameter];

display = info.display[parameter];

description = info.desc[parameter];

}

void setFromBrowseNode(BrowseNode node)

{

filename = node.GetFile();

line = node.line;

inScope = node.GetScope();

name = node.name;

kind = node.kind;

\_type = node.\_type;

deco = node.deco;

}

}

class LibraryInfos

{

alias BrowseInfo INFO;

alias BrowseNode VALUE;

this()

{

//                auto info = new LibraryInfo;

//                info.readJSON(r"m:\s\d\visuald\trunk\bin\Debug\visuald.json");

//                mInfos ~= info;

}

string[] findJSONFiles()

{

string[] files = Package.GetGlobalOptions().getJSONFiles();

auto srpSolution = queryService!(IVsSolution);

scope(exit) release(srpSolution);

auto solutionBuildManager = queryService!(IVsSolutionBuildManager)();

scope(exit) release(solutionBuildManager);

if(srpSolution && solutionBuildManager)

{

IEnumHierarchies pEnum;

if(srpSolution.GetProjectEnum(EPF\_LOADEDINSOLUTION|EPF\_MATCHTYPE, &g\_projectFactoryCLSID, &pEnum) == S\_OK)

{

scope(exit) release(pEnum);

IVsHierarchy pHierarchy;

while(pEnum.Next(1, &pHierarchy, null) == S\_OK)

{

scope(exit) release(pHierarchy);

IVsProjectCfg activeCfg;

if(solutionBuildManager.FindActiveProjectCfg(null, null, pHierarchy, &activeCfg) == S\_OK)

{

scope(exit) release(activeCfg);

if(Config cfg = qi\_cast!Config(activeCfg))

{

scope(exit) release(cfg);

cfg.addJSONFiles(files);

}

}

}

}

}

return files;

}

void updateDefinitions()

{

string[] files = findJSONFiles();

bool modified = false;

// remove files no longer found and update modified files

for(int i = 0; i < mInfos.length; )

{

int idx = arrIndex(files, mInfos[i].mFilename);

if(idx < 0)

{

mInfos = mInfos[0 .. i] ~ mInfos[i+1 .. $];

modified = true;

}

else

{

files = files[0 .. idx] ~ files[idx+1 .. $];

auto filetime = timeLastModified(mInfos[i].mFilename);

if(mInfos[i].mModified != filetime)

{

mInfos[i].readJSON(mInfos[i].mFilename);

modified = true;

}

i++;

}

}

// add new files

foreach(file; files)

{

auto info = new INFO;

if(info.readJSON(file))

{

mInfos ~= info;

modified = true;

}

}

if(modified)

mUpdateCounter++;

debug(FINDDEF) findDefinition("");

}

string[] findCompletions(string name, bool caseSensitive)

{

SearchData sd;

sd.caseSensitive = caseSensitive;

if(name.length)

sd.names ~= name;

string[] completions;

foreach(info; mInfos)

completions ~= info.findCompletions(sd);

return completions;

}

Definition[] findDefinition(string name)

{

SearchData sd;

sd.wholeWord = true;

sd.caseSensitive = true;

sd.noDupsOnSameLine = true;

if(name.length)

sd.names ~= name;

return findDefinition(sd);

}

Definition[] findDefinition(ref SearchData sd)

{

Definition[] defs;

foreach(info; mInfos)

defs ~= info.findDefinition(sd);

return defs;

}

VALUE findClass(string name, VALUE lookupScope)

{

return null;

}

INFO findInfo(string name)

{

foreach(info; mInfos)

{

string iname = getNameWithoutExt(info.mFilename);

if(icmp(name, iname) == 0)

return info;

}

return null;

}

@property int updateCounter() { return mUpdateCounter; }

INFO[] mInfos;

int mUpdateCounter;

}

class BrowseNode

{

string name;

string kind;

string deco;

private string \_type;

@property string type() { return \_type; }

int line;

BrowseNode parent;

BrowseNode[] members;

string GetFile()

{

if(parent)

return parent.GetFile();

return null;

}

string GetBase()

{

return null;

}

string[] GetInterfaces()

{

return null;

}

string GetScope()

{

if(!parent)

return null;

string pname = parent.name;

for(auto p = parent.parent; p; p = p.parent)

{

if(pname.length && p.name.length)

pname = p.name ~ "." ~ pname;

else

pname = p.name ~ pname;

}

return pname;

}

}

class ModuleBrowseNode : BrowseNode

{

string file;

override string GetFile()

{

return file;

}

}

class ClassBrowseNode : BrowseNode

{

string base;

string[] interfaces;

override string GetBase()

{

return base;

}

override string[] GetInterfaces()

{

return interfaces;

}

}

class BrowseInfo

{

string mFilename;

SysTime mModified;

BrowseNode[] mModules;

bool readJSON(string fileName)

{

LibraryInfo info = new LibraryInfo;

if(!info.readJSON(fileName))

return false;

mFilename = info.mFilename;

mModified = info.mModified;

destroy(mModules);

createModules(info);

if(Config cfg = getProjectConfig(mFilename))

{

cfg.GetProject().ClearLineChanges();

release(cfg);

}

return true;

}

static BrowseNode createNode(JSONValue[string] memberobj)

{

string kind;

if(JSONValue\* k = "kind" in memberobj)

if(k.type == JSON\_TYPE.STRING)

kind = k.str;

BrowseNode node;

if(kind == "module")

{

auto n = new ModuleBrowseNode;

if(JSONValue\* v = "file" in memberobj)

if(v.type == JSON\_TYPE.STRING)

n.file = v.str;

node = n;

if("name" !in memberobj)

node.name = stripExtension(baseName(n.file));

}

else if (kind == "class" || kind == "interface")

{

auto n = new ClassBrowseNode;

if(JSONValue\* base = "base" in memberobj)

if(base.type == JSON\_TYPE.STRING)

n.base = base.str;

if(JSONValue\* iface = "interfaces" in memberobj)

if(iface.type == JSON\_TYPE.ARRAY)

foreach(m; iface.array)

if(m.type == JSON\_TYPE.STRING)

n.interfaces ~= m.str;

node = n;

}

else

{

if(kind == "function")

{

if(JSONValue\* n = "name" in memberobj)

if(n.type == JSON\_TYPE.STRING)

if (n.str.startsWith("\_\_unittest") || n.str.startsWith("\_\_invariant"))

return null;

if(!("endline" in memberobj))

kind = "function decl";

}

node = new BrowseNode;

}

node.kind = kind;

getDeclarationInfo(node, memberobj);

return node;

}

static void removeEponymousTemplate(BrowseNode n)

{

if(n.parent && n.members.length == 1 && n.line == n.members[0].line &&

(n.kind == "template" || n.kind == n.members[0].kind))

{

if(startsWith(n.name, n.members[0].name ~ "("))

{

n.members[0].name = n.name;

foreach(ref m; n.parent.members)

if(m == n)

m = n.members[0];

}

}

}

void createModules(LibraryInfo info)

{

if(info.mModules.type == JSON\_TYPE.ARRAY)

{

JSONValue[] modules = info.mModules.array;

foreach(JSONValue mod; modules)

{

if(mod.type == JSON\_TYPE.OBJECT)

{

void iterate(JSONValue[string] object, BrowseNode parent)

{

BrowseNode node = createNode(object);

if(!node)

return;

if(parent)

{

parent.members ~= node;

node.parent = parent;

}

else

mModules ~= node;

if(JSONValue\* m = "members" in object)

if(m.type == JSON\_TYPE.ARRAY)

{

JSONValue[] members = m.array;

foreach(member; members)

if(member.type == JSON\_TYPE.OBJECT)

iterate(member.object, node);

}

removeEponymousTemplate(node);

}

iterate(mod.object, null);

}

}

}

}

// dg\_match returns:

// 0 - continue search

// 1 - stop search

// 2 - continue search, but prune subtree

bool iterateNodes(int delegate(BrowseNode node) dg\_match)

{

foreach(mod; mModules)

{

int iterate(BrowseNode node)

{

int res = dg\_match(node);

if(res == 1)

return 1;

if(res == 2)

return 0;

foreach(n; node.members)

{

res = iterate(n);

if(res > 0)

return res;

}

return 0;

}

if(iterate(mod) == 1)

return true;

}

return false;

}

Definition[] findDefinition(ref SearchData sd)

{

Definition[] defs;

int findDef(BrowseNode node)

{

if(sd.pruneSubtree(node))

return 2;

if(!isDeclarationKind(node.kind))

return 2;

if(sd.matchDefinition(node))

{

Definition def;

def.setFromBrowseNode(node);

sd.addDefinition(defs, def);

}

return 0;

}

iterateNodes(&findDef);

return defs;

}

string[] findCompletions(ref SearchData sd)

{

string[] cplts;

int findCplt(BrowseNode node)

{

if(startsWith(node.name, sd.names[0]))

{

// strip template arguments and constraint

string s = node.name;

int pos = indexOf(s, '(');

if(pos >= 0)

s = s[0..pos];

addunique(cplts, s);

}

return 0;

}

iterateNodes(&findCplt);

return cplts;

}

}

// This file is part of Visual D

//

// Visual D integrates the D programming language into Visual Studio

// Copyright (c) 2010 by Rainer Schuetze, All Rights Reserved

//

// Distributed under the Boost Software License, Version 1.0.

// See accompanying file LICENSE\_1\_0.txt or copy at <http://www.boost.org/LICENSE_1_0.txt>

module visuald.lexutil;

import std.exception;

import std.stdio;

import visuald.fileutil;

import vdc.lexer;

string getModuleDeclarationName(string fname)

{

string modname;

try

{

enum ParseState { kSpace, kModule, kIdent, kDot }

ParseState pstate = ParseState.kSpace;

Lexer lex;

int state = 0;

File file = File(fname, "r");

while(!file.eof())

{

string line = file.readln(); // File.byLine is unusable due to struct destructors not called (file never closed)

uint pos = 0;

while(pos < line.length)

{

int id;

uint prevpos = pos;

lex.scan(state, line, pos, id);

if(id == TOK\_Space || id == TOK\_Comment)

continue;

final switch(pstate)

{

case ParseState.kSpace:

if(id != TOK\_module)

return "";

pstate = ParseState.kModule;

break;

case ParseState.kModule:

if(id != TOK\_Identifier)

return "";

modname = line[prevpos .. pos].idup;

pstate = ParseState.kIdent;

break;

case ParseState.kIdent:

if(id != TOK\_dot)

return modname;

pstate = ParseState.kDot;

break;

case ParseState.kDot:

if(id != TOK\_Identifier)

return modname;

modname ~= "." ~ line[prevpos .. pos];

pstate = ParseState.kIdent;

break;

}

}

}

return "";

}

catch(Exception)

{

// not a valid file

return "";

}

}

// This file is part of Visual D

//

// Visual D integrates the D programming language into Visual Studio

// Copyright (c) 2010 by Rainer Schuetze, All Rights Reserved

//

// Distributed under the Boost Software License, Version 1.0.

// See accompanying file LICENSE\_1\_0.txt or copy at <http://www.boost.org/LICENSE_1_0.txt>

module visuald.library;

import visuald.comutil;

import visuald.logutil;

import visuald.hierutil;

import visuald.dpackage;

import visuald.dimagelist;

import visuald.intellisense;

import sdk.vsi.vsshell;

import sdk.vsi.vsshell80;

import sdk.win32.commctrl;

import std.json;

import std.conv;

import std.string;

class LibraryManager : DComObject, IVsLibraryMgr

{

Library[] mLibraries;

///////////////////////////

this()

{

mLibraries ~= newCom!Library;

}

~this()

{

Close();

}

HRESULT Close()

{

foreach(lib; mLibraries)

lib.Close();

mLibraries = mLibraries.init;

return S\_OK;

}

bool IsValidIndex(uint uIndex)

{

return uIndex < mLibraries.length;

}

//==========================================================================

// IVsLibraryMgr

HRESULT GetCount(ULONG \*pnCount)

{

mixin(LogCallMix2);

if(!pnCount)

return E\_INVALIDARG;

\*pnCount = mLibraries.length;

return S\_OK;

}

HRESULT GetLibraryAt(in ULONG uIndex, IVsLibrary \*pLibrary)

{

mixin(LogCallMix);

if(!pLibrary)

return E\_INVALIDARG;

if (!IsValidIndex(uIndex))

return E\_UNEXPECTED;

return mLibraries[uIndex].QueryInterface(&IID\_IVsLibrary, cast(void\*\*) pLibrary);

}

HRESULT GetNameAt(in ULONG uIndex, WCHAR \*\* pszName)

{

mixin(LogCallMix2);

if(!pszName)

return E\_INVALIDARG;

if (!IsValidIndex(uIndex))

return E\_UNEXPECTED;

return mLibraries[uIndex].GetName(pszName);

}

HRESULT ToggleCheckAt(in ULONG uIndex)

{

mixin(LogCallMix2);

if (!IsValidIndex(uIndex))

return E\_UNEXPECTED;

mLibraries[uIndex].ToggleCheck();

return S\_OK;

}

HRESULT GetCheckAt(in ULONG uIndex, LIB\_CHECKSTATE \*pstate)

{

mixin(LogCallMix2);

if(!pstate)

return E\_INVALIDARG;

if (!IsValidIndex(uIndex))

return E\_UNEXPECTED;

return mLibraries[uIndex].GetCheckState(pstate);

}

HRESULT SetLibraryGroupEnabled(in LIB\_PERSISTTYPE lpt, in BOOL fEnable)

{

mixin(LogCallMix2);

return E\_NOTIMPL;

}

}

class Library : DComObject,

IVsSimpleLibrary2,

IVsLiteTreeList,

//IBrowseDataProviderImpl,

//IBrowseDataProviderEvents,

IVsSolutionEvents

{

string mName = "D-Library";

LIB\_CHECKSTATE mCheckState;

HIMAGELIST mImages; //image list.

//Cookie used to hook up the solution events.

VSCOOKIE mIVsSolutionEventsCookie;

//Array of Projects

LibraryItem[] mLibraryItems;

BrowseCounter mCounterLibList;

// Find References result

string[] mLastFindReferencesResult;

override HRESULT QueryInterface(in IID\* riid, void\*\* pvObject)

{

if(\*riid == IVsLibrary2Ex.iid) // keep out of log file

return E\_NOINTERFACE;

if(queryInterface!(IVsSimpleLibrary2) (this, riid, pvObject))

return S\_OK;

//                if(queryInterface!(IVsLiteTreeList) (this, riid, pvObject))

//                        return S\_OK;

if(queryInterface!(IVsSolutionEvents) (this, riid, pvObject))

return S\_OK;

return super.QueryInterface(riid, pvObject);

}

HRESULT Initialize()

{

mixin(LogCallMix2);

mCheckState = LCS\_CHECKED;

if(auto solution = queryService!IVsSolution())

{

scope(exit) release(solution);

if(HRESULT hr = solution.AdviseSolutionEvents(this, &mIVsSolutionEventsCookie))

return hr;

}

return S\_OK;

}

HRESULT Close()

{

mixin(LogCallMix2);

if(mIVsSolutionEventsCookie != 0)

if(auto solution = queryService!IVsSolution())

{

scope(exit) release(solution);

if(HRESULT hr = solution.UnadviseSolutionEvents(mIVsSolutionEventsCookie))

return hr;

mIVsSolutionEventsCookie = 0;

}

foreach(lib; mLibraryItems)

{

lib.Close();

}

return S\_OK;

}

// ILibrary

//Return a displayable name for the designated library

HRESULT GetName(WCHAR \*\*pszName)

{

\*pszName = allocBSTR(mName);

return S\_OK;

}

//Set the selected state for a library item

HRESULT ToggleCheck()

{

mCheckState = mCheckState == LCS\_CHECKED ? LCS\_UNCHECKED : LCS\_CHECKED;

return S\_OK;

}

//Get the selected state for a library item

HRESULT GetCheckState(LIB\_CHECKSTATE \*pstate)

{

assert(pstate);

\*pstate = mCheckState;

return S\_OK;

}

HRESULT GetImageList(HANDLE \*phImageList)

{

return E\_NOTIMPL;

}

bool IsValidIndex(uint uIndex)

{

return uIndex < mLibraryItems.length;

}

HRESULT CountChecks(/\* [out] \*/ ULONG\* pcChecked, /\* [out] \*/ ULONG\* pcUnchecked)

{

assert(pcChecked);

assert(pcUnchecked);

\*pcChecked = 0;

\*pcUnchecked = 0;

foreach(lib; mLibraryItems)

{

LIB\_CHECKSTATE lcs;

lib.GetCheckState(&lcs);

if (lcs == LCS\_CHECKED)

(\*pcChecked)++;

else if (lcs == LCS\_UNCHECKED)

(\*pcUnchecked)++;

else

assert(false); // check state is not correct

}

return S\_OK;

}

// IVsSimpleLibrary2 ////////////////////////////////////////////////////////

//Return E\_FAIL if category not supported.

override HRESULT GetSupportedCategoryFields2(in LIB\_CATEGORY2 eCategory,

/+[out, retval]+/ DWORD \*pCatField)

{

mixin(LogCallMix2);

assert(pCatField);

switch(eCategory)

{

case LC\_MEMBERTYPE:

// LCMT\_METHOD = 0x0001,

// LCMT\_PROPERTY = 0x0002,

// LCMT\_EVENT = 0x0004,

// LCMT\_FIELD = 0x0008,

// LCMT\_CONSTANT = 0x0010,

// LCMT\_OPERATOR = 0x0020,

// LCMT\_MAPITEM = 0x0040,

// LCMT\_VARIABLE = 0x0080,

// LCMT\_ENUMITEM = 0x0100,

// LCMT\_TYPEDEF = 0x0200,

// LCMT\_FUNCTION = 0x0400,

\*pCatField = LCMT\_ENUMITEM | LCMT\_FUNCTION | LCMT\_VARIABLE | LCMT\_TYPEDEF | LCMT\_METHOD | LCMT\_FIELD;

break;

case LC\_MEMBERACCESS:

// LCMA\_PUBLIC = 0x0001,

// LCMA\_PRIVATE = 0x0002,

// LCMA\_PROTECTED = 0x0004,

// LCMA\_PACKAGE = 0x0008,

// LCMA\_FRIEND = 0x0010,

// LCMA\_SEALED = 0x0020

\*pCatField = LCMA\_PUBLIC; // not in JSON files

break;

case LC\_CLASSTYPE:

// LCCT\_CLASS = 0x0001,

// LCCT\_INTERFACE = 0x0002,

// LCCT\_EXCEPTION = 0x0004,

// LCCT\_STRUCT = 0x0008,

// LCCT\_ENUM = 0x0010,

// LCCT\_MODULE = 0x0020,

// LCCT\_UNION = 0x0040,

// LCCT\_INTRINSIC = 0x0080,

// LCCT\_DELEGATE = 0x0100,

// LCCT\_TYPEDEF = 0x0200,

// LCCT\_MACRO = 0x0400,

// LCCT\_MAP = 0x0800,

// LCCT\_GLOBAL = 0x1000,

\*pCatField = LCCT\_CLASS | LCCT\_INTERFACE | LCCT\_STRUCT | LCCT\_ENUM | LCCT\_MODULE | LCCT\_UNION;

break;

case LC\_CLASSACCESS:

// LCCA\_PUBLIC = 0x0001,

// LCCA\_PRIVATE = 0x0002,

// LCCA\_PROTECTED = 0x0004,

// LCCA\_PACKAGE = 0x0008,

// LCCA\_FRIEND = 0x0010,

// LCCA\_SEALED = 0x0020

\*pCatField = LCCA\_PUBLIC; // not in JSON files

break;

case LC\_ACTIVEPROJECT:

// LCAP\_SHOWALWAYS = 0x0001,

// LCAP\_MUSTBEACTIVE = 0x0002,

\*pCatField = LCAP\_SHOWALWAYS;

break;

case LC\_LISTTYPE:

// LLT\_CLASSES = 0x000001,

// LLT\_MEMBERS = 0x000002,

// LLT\_PHYSICALCONTAINERS = 0x000004,

// LLT\_PACKAGE = 0x000004, same as above (old name)

// LLT\_NAMESPACES = 0x000008,

// LLT\_CONTAINMENT = 0x000010,

// LLT\_CONTAINEDBY = 0x000020,

// LLT\_USESCLASSES = 0x000040,

// LLT\_USEDBYCLASSES = 0x000080,

// LLT\_NESTEDCLASSES = 0x000100,

// LLT\_INHERITEDINTERFACES = 0x000200,

// LLT\_INTERFACEUSEDBYCLASSES = 0x000400,

// LLT\_DEFINITIONS = 0x000800,

// LLT\_REFERENCES = 0x001000,

// LLT\_HIERARCHY = 0x002000,

\*pCatField = LLT\_NAMESPACES | LLT\_PACKAGE | LLT\_CLASSES | LLT\_MEMBERS;

break;

case LC\_VISIBILITY:

// LCV\_VISIBLE = 0x0001,

// LCV\_HIDDEN = 0x0002,

\*pCatField = LCV\_VISIBLE;

break;

case LC\_MODIFIER:

// LCMDT\_VIRTUAL = 0x0001,

// LCMDT\_PUREVIRTUAL = 0x0002,

// LCMDT\_NONVIRTUAL = 0x0004,

// LCMDT\_FINAL = 0x0008,

// LCMDT\_STATIC = 0x0010,

\*pCatField = LCMDT\_STATIC | LCMDT\_FINAL;

break;

case LC\_HIERARCHYTYPE:

\*pCatField = LCHT\_BASESANDINTERFACES;

break;

case LC\_NODETYPE:

case LC\_MEMBERINHERITANCE:

case LC\_SEARCHMATCHTYPE:

default:

\*pCatField = 0;

return E\_FAIL;

}

return S\_OK;

}

//Retrieve a IVsObjectList interface of LISTTYPE

override HRESULT GetList2(in LIB\_LISTTYPE2 eListType, in LIB\_LISTFLAGS eFlags, in VSOBSEARCHCRITERIA2 \*pobSrch,

/+[out, retval]+/ IVsSimpleObjectList2 \*ppList)

{

mixin(LogCallMix2);

//                if (eFlags & LLF\_USESEARCHFILTER)

//                        return E\_NOTIMPL;

assert(ppList);

if(pobSrch && to\_tmpwstring(pobSrch.szName) == "Find All References"w) // (pobSrch.grfOptions & VSOBSO\_LISTREFERENCES))

{

if (eListType != LLT\_MEMBERS) // also called with LLT\_NAMESPACES and LLT\_CLASSES, so avoid duplicates

return E\_FAIL;

auto frl = newCom!FindReferencesList(mLastFindReferencesResult);

return frl.QueryInterface(&IVsSimpleObjectList2.iid, cast(void\*\*) ppList);

}

else

{

auto ol = newCom!ObjectList(this, eListType, eFlags, pobSrch);

return ol.QueryInterface(&IVsSimpleObjectList2.iid, cast(void\*\*) ppList);

}

}

//Get various settings for the library

override HRESULT GetLibFlags2(/+[out, retval]+/ LIB\_FLAGS2 \*pfFlags)

{

mixin(LogCallMix2);

assert(pfFlags);

\*pfFlags = LF\_PROJECT | LF\_EXPANDABLE;

return S\_OK;

}

//Counter to check if the library has changed

override HRESULT UpdateCounter(/+[out]+/ ULONG \*pCurUpdate)

{

// mixin(LogCallMix2);

assert(pCurUpdate);

\*pCurUpdate = Package.GetLibInfos().updateCounter();

return S\_OK;

}

// Unqiue guid identifying each library that never changes (even across shell instances)

override HRESULT GetGuid(GUID\* ppguidLib)

{

mixin(LogCallMix2);

assert(ppguidLib);

\*ppguidLib = g\_omLibraryCLSID;

return S\_OK;

}

// Returns the separator string used to separate namespaces, classes and members

// eg. "::" for VC and "." for VB

override HRESULT GetSeparatorStringWithOwnership(BSTR \*pszSeparator)

{

mixin(LogCallMix2);

\*pszSeparator = allocBSTR(".");

return S\_OK;

}

//Retrieve the persisted state of this library from the passed stream

//(essentially information for each browse container being browsed). Only

//implement for GLOBAL browse containers

override HRESULT LoadState(/+[in]+/ IStream pIStream, in LIB\_PERSISTTYPE lptType)

{

mixin(LogCallMix2);

// we do not save/load persisted state

return E\_NOTIMPL;

}

//Save the current state of this library to the passed stream

//(essentially information for each browse container being browsed). Only

//implement for GLOBAL browse containers

override HRESULT SaveState(/+[in]+/ IStream pIStream, in LIB\_PERSISTTYPE lptType)

{

mixin(LogCallMix2);

// we do not save/load persisted state

return E\_NOTIMPL;

}

// Used to obtain a list of browse containers corresponding to the given

// project (hierarchy). Only return a list if your package owns this hierarchy

// Meaningful only for libraries providing PROJECT browse containers.

override HRESULT GetBrowseContainersForHierarchy(/+[in]+/ IVsHierarchy pHierarchy,

in ULONG celt,

/+[in, out, size\_is(celt)]+/ VSBROWSECONTAINER \*rgBrowseContainers,

/+[out, optional]+/ ULONG \*pcActual)

{

mixin(LogCallMix2);

if (pcActual)

\*pcActual = 0;

//Do we have this project?

foreach(lib; mLibraryItems)

{

if(lib.GetHierarchy() is pHierarchy)

{

if (celt && rgBrowseContainers)

{

rgBrowseContainers[0].pguidLib = cast(GUID\*) &g\_omLibraryCLSID;

if(HRESULT hr = lib.GetText(TTO\_DEFAULT, &rgBrowseContainers[0].szName))

return hr;

}

if (pcActual)

\*pcActual = 1;// We always only have one library.

break;

}

}

return S\_OK;

}

// Start browsing the component specified in PVSCOMPONENTSELECTORDATA (name is equivalent to that

// returned thru the liblist's GetText method for this browse container).

// Only meaningful for registered libraries for a given type of GLOBAL browse container

override HRESULT AddBrowseContainer(in PVSCOMPONENTSELECTORDATA pcdComponent,

/+[in, out]+/ LIB\_ADDREMOVEOPTIONS \*pgrfOptions,

/+[out]+/ BSTR \*pbstrComponentAdded)

{

mixin(LogCallMix2);

// we do not support GLOBAL browse containers

return E\_NOTIMPL;

}

// Stop browsing the component identified by name (name is equivalent to that

// returned thru the liblist's GetText method for this browse container

// Only meaningful for registered libraries for a given type of GLOBAL browse container

override HRESULT RemoveBrowseContainer(in DWORD dwReserved, in LPCWSTR pszLibName)

{

mixin(LogCallMix2);

// we do not support GLOBAL browse containers

return E\_NOTIMPL;

}

override HRESULT CreateNavInfo(/+[ size\_is (ulcNodes)]+/ in SYMBOL\_DESCRIPTION\_NODE \*rgSymbolNodes, in ULONG ulcNodes,

/+[out]+/ IVsNavInfo \* ppNavInfo)

{

mixin(LogCallMix2);

return E\_NOTIMPL;

}

// IVsLiteTreeList ////////////////////////////////////////////////////////

//Fetches VSTREEFLAGS

override HRESULT GetFlags(/+[out]+/ VSTREEFLAGS \*pFlags)

{

mixin(LogCallMix2);

//State change and update only

\*pFlags = TF\_NOEVERYTHING ^ (TF\_NOSTATECHANGE | TF\_NOUPDATES);

return S\_OK;

}

//Count of items in this list

override HRESULT GetItemCount(/+[out]+/ ULONG\* pCount)

{

mixin(LogCallMix2);

assert(pCount);

\*pCount = mLibraryItems.length;

return S\_OK;

}

//An item has been expanded, get the next list

override HRESULT GetExpandedList(in ULONG Index,

/+[out]+/ BOOL \*pfCanRecurse,

/+[out]+/ IVsLiteTreeList \*pptlNode)

{

mixin(LogCallMix2);

assert(\_false); // TF\_NOEXPANSION is set: this shouldn't be called

return E\_FAIL;

}

//Called during a ReAlign command if TF\_CANTRELOCATE isn't set. Return

//E\_FAIL if the list can't be located, in which case the list will be discarded.

override HRESULT LocateExpandedList(/+[in]+/ IVsLiteTreeList ExpandedList,

/+[out]+/ ULONG \*iIndex)

{

mixin(LogCallMix2);

assert(\_false); // TF\_NOEXPANSION and TF\_NORELOCATE is set: this shouldn't be called

return E\_FAIL;

}

//Called when a list is collapsed by the user.

override HRESULT OnClose(/+[out]+/ VSTREECLOSEACTIONS \*ptca)

{

mixin(LogCallMix2);

assert(ptca);

// Since handing the list back out is almost free and

// the list isn't expandable, there's no reason for

// the tree to keep a reference.

\*ptca = TCA\_CLOSEANDDISCARD;

return S\_OK;

}

//Get a pointer to the main text for the list item. Caller will NOT free, implementor

//can reuse buffer for each call to GetText except for TTO\_SORTTEXT. See VSTREETEXTOPTIONS for tto details

override HRESULT GetText(in ULONG uIndex, in VSTREETEXTOPTIONS tto,

/+[out]+/ const( WCHAR)\*\*ppszText)

{

// mixin(LogCallMix2);

assert(ppszText);

if (!IsValidIndex(uIndex))

return E\_UNEXPECTED;

return mLibraryItems[uIndex].GetText(tto, ppszText);

}

//Get a pointer to the tip text for the list item. Like GetText, caller will NOT free, implementor

//can reuse buffer for each call to GetTipText. If you want tiptext to be same as TTO\_DISPLAYTEXT, you can

//E\_NOTIMPL this call.

override HRESULT GetTipText(in ULONG uIndex, in VSTREETOOLTIPTYPE eTipType,

/+[out]+/ const( WCHAR)\*\*ppszText)

{

mixin(LogCallMix2);

assert(ppszText);

if (!IsValidIndex(uIndex))

return E\_UNEXPECTED;

return mLibraryItems[uIndex].GetTipText(eTipType, ppszText);

}

//Is this item expandable? Not called if TF\_NOEXPANSION is set

override HRESULT GetExpandable(in ULONG uIndex,

/+[out]+/ BOOL \*pfExpandable)

{

mixin(LogCallMix2);

assert(pfExpandable);

if (!IsValidIndex(uIndex))

return E\_UNEXPECTED;

\*pfExpandable = FALSE;

return S\_OK;

}

//Retrieve information to draw the item

/+[local]+/ override HRESULT GetDisplayData(in ULONG uIndex,

/+[out]+/ VSTREEDISPLAYDATA \*pData)

{

//mixin(LogCallMix2);

assert(pData);

if (!IsValidIndex(uIndex))

return E\_UNEXPECTED;

GetImageList(&pData.hImageList);

BOOL fIsLibraryChecked = (mCheckState == LCS\_UNCHECKED) ? FALSE : TRUE;

return mLibraryItems[uIndex].GetDisplayData(fIsLibraryChecked, pData);

}

//Return latest update increment. True/False isn't sufficient here since

//multiple trees may be using this list. Returning an update counter > than

//the last one cached by a given tree will force calls to GetItemCount and

//LocateExpandedList as needed.

override HRESULT UpdateCounter(/+[out]+/ ULONG \*pCurUpdate,

/+[out]+/ VSTREEITEMCHANGESMASK \*pgrfChanges)

{

// mixin(LogCallMix2);

return mCounterLibList.UpdateCounter(pCurUpdate, pgrfChanges);

}

// If prgListChanges is NULL, should return the # of changes in pcChanges. Otherwise

// \*pcChanges will indicate the size of the array (so that caller can allocate the array) to fill

// with the VSTREELISTITEMCHANGE records

override HRESULT GetListChanges(/+[in,out]+/ ULONG \*pcChanges,

/+[ size\_is (\*pcChanges)]+/ in VSTREELISTITEMCHANGE \*prgListChanges)

{

mixin(LogCallMix2);

// bad "in" in annotation of VSI SDK vsshell.h

return mCounterLibList.GetListChanges(pcChanges, cast(VSTREELISTITEMCHANGE \*)prgListChanges);

}

//Toggles the state of the given item (may be more than two states)

override HRESULT ToggleState(in ULONG uIndex,

/+[out]+/ VSTREESTATECHANGEREFRESH \*ptscr)

{

mixin(LogCallMix2);

if (!IsValidIndex(uIndex))

return E\_UNEXPECTED;

assert(ptscr);

if(HRESULT hr = mLibraryItems[uIndex].ToggleState())

return hr;

\*ptscr = TSCR\_CURRENT | TSCR\_PARENTS | TSCR\_CHILDREN | TSCR\_PARENTSCHILDREN;

LIB\_CHECKSTATE lcs;

mLibraryItems[uIndex].GetCheckState(&lcs);

// check if this change the library state

BOOL fUpdateLibraryCounter = FALSE;

ULONG cChecked;

ULONG cUnchecked;

CountChecks(&cChecked,&cUnchecked);

if (lcs == LCS\_CHECKED) // item has been checked

{

// we should update if the library has been unchecked

fUpdateLibraryCounter = (mCheckState == LCS\_UNCHECKED);

if (!cUnchecked) // the last unchecked has been checked

mCheckState = LCS\_CHECKED;// change the library state

else

// change the library state

mCheckState = LCS\_CHECKEDGRAY;

}

else // item has been unchecked

{

if (mCheckState != LCS\_UNCHECKED)

{

if (!cChecked)

{

// the last checked has been unchecked

mCheckState = LCS\_UNCHECKED;

// we should update if the library is unchecked

fUpdateLibraryCounter = TRUE;

}

else

mCheckState = LCS\_CHECKEDGRAY;

}

}

if (fUpdateLibraryCounter)

{

version(todo)

{

// notify any lists for the change

CBrowseNode \* pBrowseNode = mLibraryItem[uIndex];

if (lcs == LCS\_CHECKED)

NotifyOnBrowseDataAdded(LLT\_PACKAGE, pBrowseNode);

else

NotifyOnBrowseDataRemoved(LLT\_PACKAGE, pBrowseNode);

}

}

return S\_OK;

}

// IVsSolutionEvents //////////////////////////////////////////////////////

// fAdded == TRUE means project added to solution after solution open.

// fAdded == FALSE means project added to solution during solution open.

override HRESULT OnAfterOpenProject(/+[in]+/ IVsHierarchy pIVsHierarchy, in BOOL fAdded)

{

mixin(LogCallMix2);

assert(pIVsHierarchy);

//Do we already have this project?

foreach(lib; mLibraryItems)

if(lib.GetHierarchy() is pIVsHierarchy)

return S\_OK;

// check to see if this is a myc project

GUID guidProject;

HRESULT hr = pIVsHierarchy.GetGuidProperty(VSITEMID\_ROOT, VSHPROPID\_TypeGuid, &guidProject);

if(FAILED(hr))

return hr;

if (guidProject != g\_projectFactoryCLSID)

return S\_OK;

//Create a new project info struct

auto libraryItem = new LibraryItem(this, pIVsHierarchy);

mLibraryItems ~= libraryItem;

version(todo)

{

// inform the lists if any

NotifyOnBrowseDataAdded(LLT\_PACKAGE, pLibraryItem);

}

// update the liblist

VSTREELISTITEMCHANGE listChanges;

listChanges.grfChange = TCT\_ITEMADDED;

listChanges.Index = mLibraryItems.length - 1;

return mCounterLibList.Increment(listChanges);

}

// fRemoving == TRUE means project being removed from solution before solution close.

// fRemoving == FALSE means project being removed from solution during solution close.

override HRESULT OnQueryCloseProject(/+[in]+/ IVsHierarchy pHierarchy, in BOOL fRemoving,

/+[in,out]+/ BOOL \*pfCancel)

{

mixin(LogCallMix2);

return S\_OK;

}

// fRemoved == TRUE means project removed from solution before solution close.

// fRemoved == FALSE means project removed from solution during solution close.

override HRESULT OnBeforeCloseProject(/+[in]+/ IVsHierarchy pHierarchy, in BOOL fRemoved)

{

mixin(LogCallMix2);

assert(pHierarchy);

//Do we have this project?

int idx;

for(idx = 0; idx < mLibraryItems.length; idx++)

if(mLibraryItems[idx].GetHierarchy() is pHierarchy)

break;

if(idx >= mLibraryItems.length)

return S\_OK;

// remove the data

LibraryItem libraryItem = mLibraryItems[idx];

mLibraryItems = mLibraryItems[0..idx] ~ mLibraryItems[idx+1..$];

version(todo)

{

// inform the lists if any

NotifyOnBrowseDataRemoved(LLT\_PACKAGE, pLibraryItem);

}

// update the liblist

VSTREELISTITEMCHANGE listChanges;

listChanges.grfChange = TCT\_ITEMDELETED;

listChanges.Index = idx;

HRESULT hr = mCounterLibList.Increment(listChanges);

libraryItem.Close();

return hr;

}

// stub hierarchy is placeholder hierarchy for unloaded project.

override HRESULT OnAfterLoadProject(/+[in]+/ IVsHierarchy pStubHierarchy, /+[in]+/ IVsHierarchy pRealHierarchy)

{

mixin(LogCallMix2);

return S\_OK;

}

override HRESULT OnQueryUnloadProject(/+[in]+/ IVsHierarchy pRealHierarchy,

/+[in,out]+/ BOOL \*pfCancel)

{

mixin(LogCallMix2);

return S\_OK;

}

override HRESULT OnBeforeUnloadProject(/+[in]+/ IVsHierarchy pRealHierarchy, /+[in]+/ IVsHierarchy pStubHierarchy)

{

mixin(LogCallMix2);

return S\_OK;

}

// fNewSolution == TRUE means solution is being created now.

// fNewSolution == FALSE means solution was created previously, is being loaded.

override HRESULT OnAfterOpenSolution(/+[in]+/ IUnknown pUnkReserved, in BOOL fNewSolution)

{

mixin(LogCallMix2);

return S\_OK;

}

override HRESULT OnQueryCloseSolution(/+[in]+/ IUnknown pUnkReserved,

/+[in,out]+/ BOOL \*pfCancel)

{

mixin(LogCallMix2);

return S\_OK;

}

override HRESULT OnBeforeCloseSolution(/+[in]+/ IUnknown pUnkReserved)

{

mixin(LogCallMix2);

return S\_OK;

}

override HRESULT OnAfterCloseSolution(/+[in]+/ IUnknown pUnkReserved)

{

mixin(LogCallMix2);

return S\_OK;

}

}

enum useJSON = false;

static if(useJSON)

alias JSONValue InfoObject;

else

alias BrowseNode InfoObject;

int GetInfoCount(BrowseNode val) { return val ? val.members.length : 0; }

string GetInfoName(BrowseNode val) { return val ? val.name : null; }

string GetInfoKind(BrowseNode val) { return val ? val.kind : null; }

string GetInfoType(BrowseNode val) { return val ? val.type : null; }

string GetInfoBase(BrowseNode val) { return val ? val.GetBase() : null; }

string[] GetInfoInterfaces(BrowseNode val) { return val ? val.GetInterfaces() : null; }

string GetInfoFilename(BrowseNode val) { return val ? val.GetFile() : null; }

int GetInfoLine(BrowseNode val) { return val ? val.line : -1; }

string GetInfoScope(BrowseNode val) { return val ? val.GetScope() : null; }

BrowseNode GetInfoObject(BrowseNode val, ULONG idx)

{

if(!val || idx >= val.members.length)

return null;

return val.members[idx];

}

// move to intellisense.d?

int GetInfoCount(JSONValue val)

{

if(val.type == JSON\_TYPE.ARRAY)

return val.array.length;

if(val.type == JSON\_TYPE.OBJECT)

if(JSONValue\* m = "members" in val.object)

if(m.type == JSON\_TYPE.ARRAY)

return m.array.length;

return 0;

}

string GetInfoName(JSONValue val)

{

if(val.type == JSON\_TYPE.OBJECT)

if(JSONValue\* v = "name" in val.object)

if(v.type == JSON\_TYPE.STRING)

return v.str;

if(val.type == JSON\_TYPE.STRING)

return val.str;

return null;

}

string GetInfoKind(JSONValue val)

{

if(val.type == JSON\_TYPE.OBJECT)

if(JSONValue\* v = "kind" in val.object)

if(v.type == JSON\_TYPE.STRING)

return v.str;

if(val.type == JSON\_TYPE.STRING)

return "class";

return null;

}

string GetInfoType(JSONValue val)

{

if(val.type == JSON\_TYPE.OBJECT)

if(JSONValue\* v = "type" in val.object)

if(v.type == JSON\_TYPE.STRING)

return v.str;

return null;

}

string GetInfoBase(JSONValue val)

{

if(val.type == JSON\_TYPE.OBJECT)

if(JSONValue\* v = "base" in val.object)

if(v.type == JSON\_TYPE.STRING)

return v.str;

return null;

}

string[] GetInfoInterfaces(JSONValue val)

{

string[] ifaces;

if(val.type == JSON\_TYPE.OBJECT)

if(JSONValue\* v = "interfaces" in val.object)

if(v.type == JSON\_TYPE.ARRAY)

foreach(i, iface; v.array)

if(iface.type == JSON\_TYPE.STRING)

ifaces ~= iface.str;

return ifaces;

}

string GetInfoFilename(JSONValue val)

{

if(val.type == JSON\_TYPE.OBJECT)

if(JSONValue\* v = "file" in val.object)

if(v.type == JSON\_TYPE.STRING)

return v.str;

return null;

}

int GetInfoLine(JSONValue val)

{

if(val.type == JSON\_TYPE.OBJECT)

if(JSONValue\* v = "line" in val.object)

if(v.type == JSON\_TYPE.INTEGER)

return cast(int) v.integer - 1;

return -1;

}

JSONValue GetInfoObject(JSONValue val, ULONG idx)

{

if(val.type == JSON\_TYPE.ARRAY)

if(idx < val.array.length)

return val.array[idx];

if(val.type == JSON\_TYPE.OBJECT)

if(JSONValue\* m = "members" in val.object)

if(m.type == JSON\_TYPE.ARRAY)

if(idx < m.array.length)

return m.array[idx];

return JSONValue();

}

bool HasFunctionPrototype(string kind)

{

switch(kind)

{

case "constructor":

case "destructor":

case "allocator":

case "deallocator":

case "delegate":

case "function":

case "function decl":

return true;

default:

return false;

}

}

LIB\_LISTTYPE2 GetListType(string kind)

{

switch(kind)

{

case "union":

case "struct":

case "anonymous struct":

case "anonymous union":

case "interface":

case "enum":

case "class": return LLT\_CLASSES;

case "module": return LLT\_NAMESPACES | LLT\_HIERARCHY | LLT\_PACKAGE;

case "variable":

case "constructor":

case "destructor":

case "allocator":

case "deallocator":

case "enum member":

case "template":

case "alias":

case "typedef":

case "delegate":

case "function decl":

case "function": return LLT\_MEMBERS;

// not expected to show up in json file

case "attribute":

case "function alias":

case "alias this":

case "pragma":

case "import":

case "static import":

case "static if":

case "static assert":

case "template instance":

case "mixin":

case "debug":

case "version": return LLT\_MEMBERS;

default: return LLT\_MEMBERS;

}

}

///////////////////////////////////////////////////////////////////////

class ObjectList : DComObject, IVsSimpleObjectList2

{

// CComPtr<IBrowseDataProvider> m\_srpIBrowseDataProvider;

// VSCOOKIE m\_dwIBrowseDataProviderEventsCookie;

Library mLibrary; // Holds a pointer to the library

LIB\_LISTTYPE mListType; //type of the list

LIB\_LISTFLAGS mFlags;

const(VSOBSEARCHCRITERIA2) \*mObSrch; // assume valid through the lifetime of the list

BrowseCounter mCounter;

LibraryInfo mLibInfo;

ObjectList mParent;

InfoObject mObject;

InfoObject[] mMembers;

this(Library lib, in LIB\_LISTTYPE2 eListType, in LIB\_LISTFLAGS eFlags, in VSOBSEARCHCRITERIA2 \*pobSrch)

{

mLibrary = lib;

mListType = eListType;

mFlags = eFlags;

mObSrch = pobSrch;

initMembers();

}

this(Library lib, LibraryInfo libInfo, ObjectList parent, InfoObject object,

in LIB\_LISTTYPE2 eListType, in LIB\_LISTFLAGS eFlags, in VSOBSEARCHCRITERIA2 \*pobSrch)

{

mListType = eListType;

mFlags = eFlags;

mObSrch = pobSrch;

mLibrary = lib;

mLibInfo = libInfo;

mParent = parent;

mObject = object;

initMembers();

}

override HRESULT QueryInterface(in IID\* riid, void\*\* pvObject)

{

if(queryInterface!(IVsSimpleObjectList2) (this, riid, pvObject))

return S\_OK;

return super.QueryInterface(riid, pvObject);

}

void initMembers()

{

InfoObject[] arr;

if(!mParent)

{

// all modules from the library

auto infos = Package.GetLibInfos();

foreach(info; infos.mInfos)

static if(!useJSON)

arr ~= info.mModules;

else

if(info.mModules.type == JSON\_TYPE.ARRAY)

arr ~= info.mModules.array;

}

else if(mObject)

{

arr = mObject.members;

string base = mObject.GetBase();

string[] ifaces = mObject.GetInterfaces();

bool hasBase = base.length || ifaces.length;

// do not show base class of enums in the tree

if(mObject.kind == "enum")

hasBase = false;

if(hasBase && (mListType & LLT\_CLASSES))

{

InfoObject bc = new InfoObject;

bc.name = "Base Classes";

bc.kind = "class";

bc.parent = mObject;

bc.line = -2;

mMembers ~= bc;

void addBase(string name, string kind)

{

auto infos = Package.GetLibInfos();

InfoObject n = infos.findClass(name, mObject);

if(!n)

{

n = new InfoObject;

n.name = name;

n.kind = kind;

n.parent = bc;

n.line = -2;

}

bc.members ~= n;

}

if(base.length)

addBase(base, "class");

foreach(iface; ifaces)

addBase(iface, "interface");

}

}

string searchName;

if((mFlags & LLF\_USESEARCHFILTER) && mObSrch && mObSrch.szName)

{

searchName = to\_string(mObSrch.szName);

if(!(mObSrch.grfOptions & VSOBSO\_CASESENSITIVE))

searchName = toLower(searchName);

}

foreach(v; arr)

{

void addIfCorrectKind(InfoObject val)

{

string kind = GetInfoKind(val);

if(mListType & GetListType(kind))

mMembers ~= val;

}

if(searchName.length)

{

void searchRecurse(InfoObject val)

{

string name = GetInfoName(val);

if(!(mObSrch.grfOptions & VSOBSO\_CASESENSITIVE))

name = toLower(name);

bool rc;

switch(mObSrch.eSrchType)

{

case SO\_ENTIREWORD:

rc = (name == searchName);

break;

case SO\_SUBSTRING:

rc = (indexOf(name, searchName) >= 0);

break;

case SO\_PRESTRING:

rc = startsWith(name, searchName);

break;

default:

rc = false;

break;

}

if(rc)

addIfCorrectKind(val);

foreach(v2; val.members)

searchRecurse(v2);

}

searchRecurse(v);

}

else

addIfCorrectKind(v);

}

}

int GetCount()

{

return mMembers.length;

}

bool IsValidIndex(/\* [in] \*/ ULONG uIndex)

{

return uIndex < GetCount();

}

InfoObject GetObject(ULONG idx)

{

if(idx >= mMembers.length)

return null;

return mMembers[idx];

}

string GetName(ULONG idx)

{

InfoObject v = GetObject(idx);

return GetInfoName(v);

}

string GetKind(ULONG idx)

{

InfoObject v = GetObject(idx);

return GetInfoKind(v);

}

InfoObject GetModule()

{

if(GetInfoKind(mObject) == "module")

return mObject;

if(mParent)

return mParent.GetModule();

return null;

}

InfoObject GetModule(ULONG idx)

{

if(GetKind(idx) == "module")

return GetObject(idx);

return GetModule();

}

// IVsLiteTreeList ///////////////////////////////////////////////////////

override HRESULT GetFlags(/+[out]+/ VSTREEFLAGS \*pFlags)

{

mixin(LogCallMix2);

assert(pFlags);

if (GetCount() > 0) // mListType & (LLT\_PACKAGE | LLT\_CLASSES))

{

//State change and expansion

\*pFlags = TF\_NOEVERYTHING ^ (TF\_NOSTATECHANGE | TF\_NOUPDATES | TF\_NOEXPANSION);

}

else

{

//State change only

\*pFlags = TF\_NOEVERYTHING ^ (TF\_NOSTATECHANGE | TF\_NOUPDATES);

}

return S\_OK;

}

//Count of items in this list

override HRESULT GetItemCount(/+[out]+/ ULONG\* pCount)

{

mixin(LogCallMix2);

assert(pCount);

\*pCount = GetCount();

return S\_OK;

}

//Called when a list is collapsed by the user.

override HRESULT OnClose(/+[out]+/ VSTREECLOSEACTIONS \*ptca)

{

mixin(LogCallMix2);

assert(ptca);

\*ptca = TCA\_NOTHING;

return E\_NOTIMPL;

}

override HRESULT GetTextWithOwnership(in ULONG uIndex, in VSTREETEXTOPTIONS tto,

/+[out]+/ BSTR \*pbstrText)

{

//mixin(LogCallMix2);

if (!IsValidIndex(uIndex))

return E\_UNEXPECTED;

auto val = GetObject(uIndex);

string name = GetInfoName(val);

if(mFlags & LLF\_USESEARCHFILTER)

{

string scp = GetInfoScope(val);

if(scp.length)

name = scp ~ "." ~ name;

}

Definition def;

if(val)

def.setFromBrowseNode(val);

if(HasFunctionPrototype(def.kind))

{

string ret = def.GetReturnType();

name = ret ~ " " ~ name ~ "(";

for(int i = 0; i < def.GetParameterCount(); i++)

{

string pname, description, display;

def.GetParameterInfo(i, pname, display, description);

if(i > 0)

name ~= ", ";

name ~= display;

}

name ~= ")";

}

\*pbstrText = allocBSTR(name);

return S\_OK;

}

//If you want tiptext to be same as TTO\_DISPLAYTEXT, you can E\_NOTIMPL this call.

override HRESULT GetTipTextWithOwnership(in ULONG uIndex, in VSTREETOOLTIPTYPE eTipType,

/+[out]+/ BSTR \*pbstrText)

{

mixin(LogCallMix2);

if (!IsValidIndex(uIndex))

return E\_UNEXPECTED;

string kind = GetKind(uIndex);

string name = GetName(uIndex);

\*pbstrText = allocBSTR(kind ~ " " ~ name);

return S\_OK;

}

//Retrieve information to draw the item

/+[local]+/ HRESULT GetDisplayData(in ULONG Index, /+[out]+/ VSTREEDISPLAYDATA \*pData)

{

//mixin(LogCallMix2);

pData.Mask = TDM\_IMAGE | TDM\_SELECTEDIMAGE;

string kind = GetKind(Index);

switch(kind)

{

case "class": pData.Image = CSIMG\_CLASS; break;

case "module": pData.Image = CSIMG\_PACKAGE; break;

case "variable": pData.Image = CSIMG\_FIELD; break;

case "constructor":

case "destructor":

case "allocator":

case "deallocator":

case "function decl":

case "function": pData.Image = CSIMG\_MEMBER; break;

case "delegate": pData.Image = CSIMG\_MEMBER; break;

case "interface": pData.Image = CSIMG\_INTERFACE; break;

case "union": pData.Image = CSIMG\_UNION; break;

case "struct": pData.Image = CSIMG\_STRUCT; break;

case "anonymous struct": pData.Image = CSIMG\_STRUCT; break;

case "anonymous union": pData.Image = CSIMG\_UNION; break;

case "enum": pData.Image = CSIMG\_ENUM; break;

case "enum member": pData.Image = CSIMG\_ENUMMEMBER; break;

case "template": pData.Image = CSIMG\_TEMPLATE; break;

case "alias":

case "typedef": pData.Image = CSIMG\_UNKNOWN7; break;

// not expected to show up in json file

case "attribute":

case "function alias":

case "alias this":

case "pragma":

case "import":

case "static import":

case "static if":

case "static assert":

case "template instance":

case "mixin":

case "debug":

case "version":

pData.Image = CSIMG\_BLITZ;

break;

default:

pData.Image = CSIMG\_STOP;

}

pData.SelectedImage = pData.Image;

return S\_OK;

}

//Return latest update increment. True/False isn't sufficient here since

//multiple trees may be using this list. Returning an update counter > than

//the last one cached by a given tree will force calls to GetItemCount and

//LocateExpandedList as needed.

override HRESULT UpdateCounter(/+[out]+/ ULONG \*pCurUpdate)

{

// mixin(LogCallMix2);

return mCounter.UpdateCounter(pCurUpdate, null);

}

// IVsObjectList /////////////////////////////////////////////////////////////

override HRESULT GetCapabilities2(/+[out]+/ LIB\_LISTCAPABILITIES \*pCapabilities)

{

mixin(LogCallMix2);

\*pCapabilities = LLC\_NONE;

return S\_OK;

}

// Get a sublist

override HRESULT GetList2(in ULONG uIndex, in LIB\_LISTTYPE2 ListType, in LIB\_LISTFLAGS Flags, in VSOBSEARCHCRITERIA2 \*pobSrch,

/+[out]+/ IVsSimpleObjectList2 \*ppList)

{

mixin(LogCallMix2);

auto obj = GetObject(uIndex);

if(!obj)

return E\_UNEXPECTED;

auto list = newCom!ObjectList(mLibrary, mLibInfo, this, obj, ListType, Flags, pobSrch);

return list.QueryInterface(&IVsSimpleObjectList2.iid, cast(void\*\*) ppList);

}

override HRESULT GetCategoryField2(in ULONG uIndex, in LIB\_CATEGORY2 Category,

/+[out,retval]+/ DWORD\* pField)

{

mixin(LogCallMix2);

assert(pField);

if(Category == LC\_LISTTYPE && uIndex == BrowseCounter.NULINDEX)

{

// child list types supported under this list

\*pField = LLT\_NAMESPACES | LLT\_CLASSES | LLT\_MEMBERS;

return S\_OK;

}

if (!IsValidIndex(uIndex))

return E\_UNEXPECTED;

\*pField = 0;

switch (Category)

{

case LC\_LISTTYPE:

switch (GetKind(uIndex))

{

case "module":

\*pField = LLT\_NAMESPACES | LLT\_CLASSES | LLT\_MEMBERS;

break;

case "class":

case "interface":

\*pField = LLT\_CLASSES | LLT\_MEMBERS | LLT\_HIERARCHY;

break;

case "union":

case "struct":

case "anonymous struct":

case "anonymous union":

case "enum":

\*pField = LLT\_CLASSES | LLT\_MEMBERS;

break;

default:

\*pField = 0;

break;

}

break;

case LC\_VISIBILITY:

\*pField = LCV\_VISIBLE;

break;

case LC\_MEMBERTYPE:

assert(uIndex != BrowseCounter.NULINDEX);

return E\_NOTIMPL; // m\_rgpBrowseNode[uIndex]->GetCategoryField(eCategory, pField);

case LC\_HIERARCHYTYPE:

switch (GetKind(uIndex))

{

case "class":

case "interface":

\*pField = LLT\_CLASSES | LLT\_MEMBERS | LLT\_HIERARCHY;

break;

default:

\*pField = 0;

return E\_FAIL;

}

break;

case LC\_NODETYPE:

case LC\_MEMBERINHERITANCE:

case LC\_SEARCHMATCHTYPE:

default:

\*pField = 0;

return E\_FAIL;

}

return S\_OK;

}

override HRESULT GetExpandable3(in ULONG Index, in LIB\_LISTTYPE2 ListTypeExcluded,

/+[out]+/ BOOL \*pfExpandable)

{

//mixin(LogCallMix2);

assert(pfExpandable);

InfoObject obj = GetObject(Index);

if(GetInfoCount(obj) > 0) // mListType & (LLT\_PACKAGE | LLT\_CLASSES))

\*pfExpandable = TRUE;

else

\*pfExpandable = FALSE;

return S\_OK;

}

override HRESULT GetNavInfo(in ULONG uIndex, /+[out]+/ IVsNavInfo \* ppNavInfo)

{

mixin(LogCallMix2);

return E\_NOTIMPL;

}

override HRESULT GetNavInfoNode(in ULONG Index,

/+[out]+/ IVsNavInfoNode \* ppNavInfoNode)

{

mixin(LogCallMix2);

return E\_NOTIMPL;

}

override HRESULT LocateNavInfoNode(/+[in]+/ IVsNavInfoNode pNavInfoNode,

/+[out]+/ ULONG \* pulIndex)

{

mixin(LogCallMix2);

return E\_NOTIMPL;

}

override HRESULT GetBrowseObject(in ULONG Index,

/+[out]+/ IDispatch \*ppdispBrowseObj)

{

mixin(LogCallMix2);

return E\_NOTIMPL;

}

override HRESULT GetUserContext(in ULONG Index,

/+[out]+/ IUnknown \*ppunkUserCtx)

{

mixin(LogCallMix2);

return E\_NOTIMPL;

}

override HRESULT ShowHelp(in ULONG Index)

{

mixin(LogCallMix2);

return E\_NOTIMPL;

}

override HRESULT GetSourceContextWithOwnership(in ULONG Index,

/+[out]+/ BSTR \*pszFileName,

/+[out]+/ ULONG \*pulLineNum)

{

mixin(LogCallMix2);

return E\_NOTIMPL;

}

override HRESULT GetProperty(in ULONG Index, in VSOBJLISTELEMPROPID propid,

/+[out]+/ VARIANT \*pvar)

{

mixin(LogCallMix2);

return E\_NOTIMPL;

}

// Returns the count of itemids (these must be from a single hierarchy) that make up the source files

// for the list element at Index. Also returns the hierarchy ptr and itemid if requested.

// If there are >1 itemids, return VSITEMID\_SELECTION and a subsequent call will be made

// on GetMultipleSourceItems to get them. If there are no available source items, return

// VSITEMID\_ROOT to indicate the root of the hierarchy as a whole.

override HRESULT CountSourceItems(in ULONG Index,

/+[out]+/ IVsHierarchy \*ppHier,

/+[out]+/ VSITEMID \*pitemid,

/+[out, retval]+/ ULONG \*pcItems)

{

mixin(LogCallMix2);

return E\_NOTIMPL;

}

// Used if CountSourceItems returns > 1. Details for filling up these out params are same

// as IVsMultiItemSelect::GetSelectedItems

override HRESULT GetMultipleSourceItems(in ULONG Index, in VSGSIFLAGS grfGSI, in ULONG cItems,

/+[out, size\_is(cItems)]+/ VSITEMSELECTION \*rgItemSel)

{

mixin(LogCallMix2);

return E\_NOTIMPL;

}

// Return TRUE if navigation to source of the specified type (definition or declaration),

// is possible, FALSE otherwise

override HRESULT CanGoToSource(in ULONG Index, in VSOBJGOTOSRCTYPE SrcType,

/+[out]+/ BOOL \*pfOK)

{

mixin(LogCallMix2);

if(SrcType != GS\_ANY)

{

if(SrcType == GS\_DEFINITION && GetKind(Index) == "function decl")

return E\_FAIL;

if(SrcType == GS\_DECLARATION && GetKind(Index) != "function decl")

return E\_FAIL;

if(SrcType != GS\_DECLARATION && SrcType != GS\_DEFINITION)

return E\_FAIL;

}

\*pfOK = TRUE;

return S\_OK;

}

// Called to cause navigation to the source (definition or declration) for the

// item Index. You must must coordinate with the project system to open the

// source file and navigate it to the approp. line. Return S\_OK on success or an

// hr error (along with rich error info if possible) if the navigation failed.

override HRESULT GoToSource(in ULONG Index, in VSOBJGOTOSRCTYPE SrcType)

{

mixin(LogCallMix2);

string file, modname;

auto mod = GetModule(Index);

if(mod)

{

file = GetInfoFilename(mod);

modname = GetInfoName(mod);

}

auto obj = GetObject(Index);

int line = GetInfoLine(obj);

if(file.length == 0)

file = GetInfoFilename(obj);

return OpenFileInSolution(file, line, 0, modname, true);

}

override HRESULT GetContextMenu(in ULONG Index,

/+[out]+/ CLSID \*pclsidActive,

/+[out]+/ LONG \*pnMenuId,

/+[out]+/ IOleCommandTarget \*ppCmdTrgtActive)

{

// mixin(LogCallMix2);

return E\_NOTIMPL;

}

override HRESULT QueryDragDrop(in ULONG Index, /+[in]+/ IDataObject pDataObject, in DWORD grfKeyState,

/+[in, out]+/DWORD \* pdwEffect)

{

mixin(LogCallMix2);

return E\_NOTIMPL;

}

override HRESULT DoDragDrop(in ULONG Index, /+[in]+/ IDataObject pDataObject, in DWORD grfKeyState,

/+[in, out]+/DWORD \* pdwEffect)

{

mixin(LogCallMix2);

return E\_NOTIMPL;

}

// Says whether the item Index can be renamed or not. If the passed in pszNewName is NULL,

// it simply answers the general question of whether or not that item supports rename

// (return TRUE or FALSE). If pszNewName is non-NULL, do validation of the new name

// and return TRUE if successful rename with that new name is possible or an an error hr (along with FALSE)

// if the name is somehow invalid (and set the rich error info to indicate to the user

// what was wrong)

override HRESULT CanRename(in ULONG Index, in LPCOLESTR pszNewName,

/+[out]+/ BOOL \*pfOK)

{

mixin(LogCallMix2);

return E\_NOTIMPL;

}

// Called when the user commits the Rename operation. Guaranteed that CanRename has already

// been called with the newname so that you've had a chance to validate the name. If

// Rename succeeds, return S\_OK, other wise error hr (and set the rich error info)

// indicating the problem encountered.

override HRESULT DoRename(in ULONG Index, in LPCOLESTR pszNewName, in VSOBJOPFLAGS grfFlags)

{

mixin(LogCallMix2);

return E\_NOTIMPL;

}

// Says whether the item Index can be deleted or not. Return TRUE if it can, FALSE if not.

override HRESULT CanDelete(in ULONG Index,

/+[out]+/ BOOL \*pfOK)

{

mixin(LogCallMix2);

return E\_NOTIMPL;

}

// Called when the user asks to delete the item at Index. Will only happen if CanDelete on

// the item previously returned TRUE. On a successful deletion this should return S\_OK, if

// the deletion failed, return the failure as an error hresult and set any pertinent error

// info in the standard ole error info.

override HRESULT DoDelete(in ULONG Index, in VSOBJOPFLAGS grfFlags)

{

mixin(LogCallMix2);

return E\_NOTIMPL;

}

// Used to add the description pane text in OBject Browser. Also an alternate

// mechanism for providing tooltips (ODO\_TOOLTIPDESC is set in that case)

override HRESULT FillDescription2(in ULONG Index, in VSOBJDESCOPTIONS grfOptions, /+[in]+/ IVsObjectBrowserDescription3 pobDesc)

{

mixin(LogCallMix2);

auto val = GetObject(Index);

Definition def;

if(val)

def.setFromBrowseNode(val);

if(!val || def.line < -1)

return S\_OK; // no description for auto generated nodes

if(HasFunctionPrototype(def.kind))

{

string ret = def.GetReturnType();

pobDesc.AddDescriptionText3(\_toUTF16z(ret), OBDS\_TYPE, null);

pobDesc.AddDescriptionText3(" ", OBDS\_MISC, null);

pobDesc.AddDescriptionText3(\_toUTF16z(def.name), OBDS\_NAME, null);

pobDesc.AddDescriptionText3("(", OBDS\_MISC, null);

for(int i = 0; i < def.GetParameterCount(); i++)

{

string name, description, disp;

def.GetParameterInfo(i, name, disp, description);

if(i > 0)

pobDesc.AddDescriptionText3(", ", OBDS\_COMMA, null);

pobDesc.AddDescriptionText3(\_toUTF16z(disp), OBDS\_PARAM, null);

}

pobDesc.AddDescriptionText3(")\n", OBDS\_MISC, null);

}

else

{

pobDesc.AddDescriptionText3("Name: ", OBDS\_MISC, null);

pobDesc.AddDescriptionText3(\_toUTF16z(def.name), OBDS\_NAME, null);

if(def.type.length)

{

pobDesc.AddDescriptionText3("\nType: ", OBDS\_MISC, null);

pobDesc.AddDescriptionText3(\_toUTF16z(def.type), OBDS\_TYPE, null);

}

if(def.kind.length)

{

pobDesc.AddDescriptionText3("\nKind: ", OBDS\_MISC, null);

pobDesc.AddDescriptionText3(\_toUTF16z(def.kind), OBDS\_TYPE, null);

}

string base = GetInfoBase(val);

if(base.length)

{

pobDesc.AddDescriptionText3("\nBase: ", OBDS\_MISC, null);

pobDesc.AddDescriptionText3(\_toUTF16z(base), OBDS\_TYPE, null);

}

string[] ifaces = GetInfoInterfaces(val);

if(ifaces.length)

{

pobDesc.AddDescriptionText3("\nInterfaces: ", OBDS\_MISC, null);

foreach(i, iface; ifaces)

{

if(i > 0)

pobDesc.AddDescriptionText3(", ", OBDS\_MISC, null);

pobDesc.AddDescriptionText3(\_toUTF16z(iface), OBDS\_TYPE, null);

}

}

}

string filename = GetInfoFilename(GetModule(Index));

if(filename.length == 0)

filename = GetInfoFilename(val);

if(filename.length)

{

string msg = "\n\nFile: " ~ filename;

pobDesc.AddDescriptionText3(\_toUTF16z(msg), OBDS\_MISC, null);

if(def.line >= 0)

{

msg = "(" ~ to!string(def.line) ~ ")";

pobDesc.AddDescriptionText3(\_toUTF16z(msg), OBDS\_MISC, null);

}

}

return S\_OK;

}

// These three methods give the list a chance to provide clipboard formats for a drag-drop or

// copy/paste operation.

// Caller first calls EnumClipboardFormats(index, flags, 0, NULL, &cExpected) to get the count

// of clipboard formats the list is interested in providing, allocates an array of that size,

// and then calls EnumClipboardFormats(index, flags, cExpected, prgCFs, &cActual)

// Flags indicate whether this is part of a multiple selction of items. In the

// returned array, the list can indicate which formats it supports, on what STGMEDIUM and

// whether the format is a composite one (caller does the actual rendering after calling

// GetExtendedClipboardVariant) vs one that the list itself will render thru GetClipboardFormat

// In the case of a multi-select, typically the list would only support composite formats

// enabling the caller to write the format in the form:

// <count of items><foo variant from selected item1><foo variant from selected item2>..

// (Note that only certain persistable VARIANT types are supported (as per CComVariant::WriteToStream).

// In the single select case, the list is free to provide both traditional and composite formats

// and will be called respectively on GetClipboardFormat or GetExtendedClipboardVariant for each.

// Note that CV/OB will automatically provide a CF\_NAVINFO and a CF\_TEXT/CF\_UNICODETEXT format, so

// EnumClipboardFormats should NOT return these values.

override HRESULT EnumClipboardFormats(in ULONG Index,

in VSOBJCFFLAGS grfFlags,

in ULONG celt,

/+[in, out, size\_is(celt)]+/ VSOBJCLIPFORMAT \*rgcfFormats,

/+[out, optional]+/ ULONG \*pcActual)

{

mixin(LogCallMix2);

return E\_NOTIMPL;

}

override HRESULT GetClipboardFormat(in ULONG Index,

in VSOBJCFFLAGS grfFlags,

in FORMATETC \*pFormatetc,

in STGMEDIUM \*pMedium)

{

mixin(LogCallMix2);

return E\_NOTIMPL;

}

override HRESULT GetExtendedClipboardVariant(in ULONG Index,

in VSOBJCFFLAGS grfFlags,

in const( VSOBJCLIPFORMAT)\*pcfFormat,

/+[out]+/ VARIANT \*pvarFormat)

{

mixin(LogCallMix2);

return E\_NOTIMPL;

}

}

class FindReferencesList : DComObject, IVsSimpleObjectList2

{

string[] mReferences;

this(string[] refs)

{

mReferences = refs;

}

override HRESULT QueryInterface(in IID\* riid, void\*\* pvObject)

{

if(queryInterface!(IVsSimpleObjectList2) (this, riid, pvObject))

return S\_OK;

return super.QueryInterface(riid, pvObject);

}

bool IsValidIndex(uint uIndex)

{

return uIndex < mReferences.length;

}

string getSourceLoc(uint Index, int\* line = null, int \*col = null)

{

if (!IsValidIndex(Index))

return null;

string r = mReferences[Index];

auto idx = indexOf(r, ':');

if(idx > 0)

{

string[] num = split(r[0..idx], ",");

if(num.length == 4)

{

try

{

if(line)

\*line = parse!int(num[0]) - 1;

if(col)

\*col = parse!int(num[1]);

return r[idx+1..$];

}

catch(ConvException)

{

}

}

}

return null;

}

///////////////////////////////////////////////////////////////////

HRESULT GetFlags(/+[out]+/ VSTREEFLAGS \*pFlags)

{

mixin(LogCallMix2);

return E\_NOTIMPL;

}

HRESULT GetCapabilities2(/+[out]+/ LIB\_LISTCAPABILITIES2 \*pgrfCapabilities)

{

mixin(LogCallMix2);

return E\_NOTIMPL;

}

HRESULT UpdateCounter(/+[out]+/ ULONG \*pCurUpdate)

{

mixin(LogCallMix2);

return E\_NOTIMPL;

}

HRESULT GetItemCount(/+[out]+/ ULONG\* pCount)

{

mixin(LogCallMix2);

\*pCount = mReferences.length;

return S\_OK;

}

/+[local]+/ HRESULT GetDisplayData(in ULONG Index,

/+[out]+/ VSTREEDISPLAYDATA \*pData)

{

mixin(LogCallMix2);

if (!IsValidIndex(Index))

return E\_UNEXPECTED;

pData.Mask = TDM\_IMAGE | TDM\_SELECTEDIMAGE;

pData.Image = CSIMG\_BLITZ;

pData.SelectedImage = pData.Image;

return S\_OK;

}

HRESULT GetTextWithOwnership(in ULONG Index, in VSTREETEXTOPTIONS tto,

/+[out]+/ BSTR \*pbstrText)

{

mixin(LogCallMix2);

switch(tto)

{

case TTO\_DEFAULT:

case TTO\_SORTTEXT:

case TTO\_SEARCHTEXT:

int line, col;

string file = getSourceLoc(Index, &line, &col);

file ~= "(" ~ to!string(line) ~ "," ~ to!string(col) ~ ")";

\*pbstrText = allocBSTR(file);

return S\_OK;

default:

break;

}

return E\_FAIL;

}

HRESULT GetTipTextWithOwnership(in ULONG Index, in VSTREETOOLTIPTYPE eTipType,

/+[out]+/ BSTR \*pbstrText)

{

mixin(LogCallMix2);

if (!IsValidIndex(Index))

return E\_UNEXPECTED;

\*pbstrText = allocBSTR(mReferences[Index]);

return S\_OK;

}

HRESULT GetCategoryField2(in ULONG Index, in LIB\_CATEGORY2 Category,

/+[out,retval]+/ DWORD \*pfCatField)

{

mixin(LogCallMix2);

return E\_NOTIMPL;

}

HRESULT GetBrowseObject(in ULONG Index,

/+[out]+/ IDispatch \*ppdispBrowseObj)

{

mixin(LogCallMix2);

return E\_NOTIMPL;

}

HRESULT GetUserContext(in ULONG Index,

/+[out]+/ IUnknown \*ppunkUserCtx)

{

mixin(LogCallMix2);

return E\_NOTIMPL;

}

HRESULT ShowHelp(in ULONG Index)

{

mixin(LogCallMix2);

return E\_NOTIMPL;

}

HRESULT GetSourceContextWithOwnership(in ULONG Index,

/+[out]+/ BSTR \*pbstrFileName,

/+[out]+/ ULONG \*pulLineNum)

{

mixin(LogCallMix2);

version(none)

{

int line;

string file = getSourceLoc(Index, &line);

if (!file)

return E\_FAIL;

\*pbstrFileName = allocBSTR(file);

\*pulLineNum = line;

return S\_OK;

}

else

return E\_NOTIMPL;

}

HRESULT CountSourceItems(in ULONG Index,

/+[out]+/ IVsHierarchy \*ppHier,

/+[out]+/ VSITEMID \*pitemid,

/+[out, retval]+/ ULONG \*pcItems)

{

mixin(LogCallMix2);

return E\_NOTIMPL;

}

HRESULT GetMultipleSourceItems(in ULONG Index, in VSGSIFLAGS grfGSI, in ULONG cItems,

/+[out, size\_is(cItems)]+/ VSITEMSELECTION \*rgItemSel)

{

mixin(LogCallMix2);

return E\_NOTIMPL;

}

HRESULT CanGoToSource(in ULONG Index, in VSOBJGOTOSRCTYPE SrcType,

/+[out]+/ BOOL \*pfOK)

{

mixin(LogCallMix2);

if (!IsValidIndex(Index) || !pfOK)

return E\_UNEXPECTED;

\*pfOK = (SrcType == GS\_ANY || SrcType == GS\_REFERENCE);

return S\_OK;

}

HRESULT GoToSource(in ULONG Index, in VSOBJGOTOSRCTYPE SrcType)

{

mixin(LogCallMix2);

int line, col;

string file = getSourceLoc(Index, &line, &col);

string modname;

if(!file.length)

return E\_FAIL;

return OpenFileInSolution(file, line, col, modname, true);

}

HRESULT GetContextMenu(in ULONG Index,

/+[out]+/ CLSID \*pclsidActive,

/+[out]+/ LONG \*pnMenuId,

/+[out]+/ IOleCommandTarget \*ppCmdTrgtActive)

{

mixin(LogCallMix2);

return E\_NOTIMPL;

}

HRESULT QueryDragDrop(in ULONG Index, /+[in]+/ IDataObject pDataObject, in DWORD grfKeyState,

/+[in, out]+/DWORD \* pdwEffect)

{

mixin(LogCallMix2);

return E\_NOTIMPL;

}

HRESULT DoDragDrop(in ULONG Index, /+[in]+/ IDataObject pDataObject, in DWORD grfKeyState,

/+[in, out]+/DWORD \* pdwEffect)

{

mixin(LogCallMix2);

return E\_NOTIMPL;

}

HRESULT CanRename(in ULONG Index, in LPCOLESTR pszNewName,

/+[out]+/ BOOL \*pfOK)

{

mixin(LogCallMix2);

return E\_NOTIMPL;

}

HRESULT DoRename(in ULONG Index, in LPCOLESTR pszNewName, in VSOBJOPFLAGS grfFlags)

{

mixin(LogCallMix2);

return E\_NOTIMPL;

}

HRESULT CanDelete(in ULONG Index,

/+[out]+/ BOOL \*pfOK)

{

mixin(LogCallMix2);

return E\_NOTIMPL;

}

HRESULT DoDelete(in ULONG Index, in VSOBJOPFLAGS grfFlags)

{

mixin(LogCallMix2);

return E\_NOTIMPL;

}

HRESULT FillDescription2(in ULONG Index, in VSOBJDESCOPTIONS grfOptions, /+[in]+/ IVsObjectBrowserDescription3 pobDesc)

{

mixin(LogCallMix2);

return E\_NOTIMPL;

}

HRESULT EnumClipboardFormats(in ULONG Index, in VSOBJCFFLAGS grfFlags, in ULONG celt,

/+[in, out, size\_is(celt)]+/ VSOBJCLIPFORMAT \*rgcfFormats,

/+[out, optional]+/ ULONG \*pcActual)

{

mixin(LogCallMix2);

return E\_NOTIMPL;

}

HRESULT GetClipboardFormat(in ULONG Index, in VSOBJCFFLAGS grfFlags, in FORMATETC \*pFormatetc, in STGMEDIUM \*pMedium)

{

mixin(LogCallMix2);

return E\_NOTIMPL;

}

HRESULT GetExtendedClipboardVariant(in ULONG Index, in VSOBJCFFLAGS grfFlags, in const( VSOBJCLIPFORMAT)\*pcfFormat,

/+[out]+/ VARIANT \*pvarFormat)

{

mixin(LogCallMix2);

return E\_NOTIMPL;

}

HRESULT GetProperty(in ULONG Index, in VSOBJLISTELEMPROPID propid,

/+[out]+/ VARIANT \*pvar)

{

mixin(LogCallMix2);

return E\_NOTIMPL;

}

HRESULT GetNavInfo(in ULONG Index,

/+[out]+/ IVsNavInfo \* ppNavInfo)

{

mixin(LogCallMix2);

return E\_NOTIMPL;

}

HRESULT GetNavInfoNode(in ULONG Index,

/+[out]+/ IVsNavInfoNode \* ppNavInfoNode)

{

mixin(LogCallMix2);

return E\_NOTIMPL;

}

HRESULT LocateNavInfoNode(/+[in]+/ IVsNavInfoNode pNavInfoNode,

/+[out]+/ ULONG \* pulIndex)

{

mixin(LogCallMix2);

return E\_NOTIMPL;

}

HRESULT GetExpandable3(in ULONG Index, in LIB\_LISTTYPE2 ListTypeExcluded,

/+[out]+/ BOOL \*pfExpandable)

{

mixin(LogCallMix2);

return E\_NOTIMPL;

}

HRESULT GetList2(in ULONG Index, in LIB\_LISTTYPE2 ListType, in LIB\_LISTFLAGS Flags, in VSOBSEARCHCRITERIA2 \*pobSrch,

/+[out, retval]+/ IVsSimpleObjectList2 \*ppIVsSimpleObjectList2)

{

mixin(LogCallMix2);

return E\_NOTIMPL;

}

HRESULT OnClose(/+[out]+/ VSTREECLOSEACTIONS \*ptca)

{

mixin(LogCallMix2);

return E\_NOTIMPL;

}

};

class LibraryItem

{

this(Library lib, IVsHierarchy pIVsHierarchy)

{

mLibrary = lib;

mHierarchy = addref(pIVsHierarchy);

}

IVsHierarchy GetHierarchy() { return mHierarchy; }

void Close()

{

mHierarchy = release(mHierarchy);

}

HRESULT ToggleState()

{

mCheckState = mCheckState == LCS\_CHECKED ? LCS\_UNCHECKED : LCS\_CHECKED;

return S\_OK;

}

HRESULT GetCheckState(/\* [out] \*/ LIB\_CHECKSTATE \*pstate)

{

assert(pstate);

\*pstate = mCheckState;

return S\_OK;

}

// Get a pointer to the main text for the list

HRESULT GetText(/\* [in] \*/ VSTREETEXTOPTIONS tto, /\* [out] \*/ const(WCHAR) \*\*ppszText)

{

\*ppszText = "LibItem.GetText"w.ptr;

return S\_OK;

}

HRESULT GetTipText(in VSTREETOOLTIPTYPE eTipType, const( WCHAR)\*\*ppszText)

{

\*ppszText = "LibItem.GetTipText"w.ptr;

return S\_OK;

}

HRESULT GetDisplayData(/\* [in] \*/ BOOL fIsLibraryChecked,

/\* [out] \*/ VSTREEDISPLAYDATA \* pData)

{

assert(pData);

return E\_NOTIMPL;

}

Library mLibrary;

IVsHierarchy mHierarchy;

LIB\_CHECKSTATE mCheckState;

}

struct BrowseCounter

{

enum DWORD NULINDEX = ~0;

HRESULT ResetChanges()

{

m\_cChanges = 0;

m\_fIsCounterDirty = FALSE;

m\_listChanges.Index = NULINDEX;

m\_listChanges.grfChange = TCT\_NOCHANGE;

return S\_OK;

}

HRESULT Increment(/\* [in] \*/ VSTREELISTITEMCHANGE listChanges)

{

if (m\_fIsCounterDirty)

{

m\_listChanges.Index = NULINDEX;

m\_listChanges.grfChange = TCT\_TOOMANYCHANGES;

}

else

{

m\_listChanges = listChanges;

}

m\_fIsCounterDirty = TRUE;

m\_uCounter ++;

m\_cChanges ++;

return S\_OK;

}

HRESULT UpdateCounter(

/\* [out] \*/ ULONG \* puCurUpdate,

/\* [out] \*/ VSTREEITEMCHANGESMASK \* pgrfChanges)

{

if(puCurUpdate)

\*puCurUpdate = m\_uCounter;

if(pgrfChanges)

\*pgrfChanges = m\_listChanges.grfChange;

return S\_OK;

}

HRESULT GetListChanges(/\*[in,out] \*/ ULONG \* pcChanges,

/\*[in, size\_is(\*pcChanges)]\*/ VSTREELISTITEMCHANGE \* prgListChanges)

{

assert(pcChanges);

assert(m\_cChanges == 1);

assert(m\_fIsCounterDirty);

assert((m\_listChanges.Index != NULINDEX) ||

(m\_listChanges.grfChange == TCT\_TOOMANYCHANGES) );

assert(m\_listChanges.grfChange != TCT\_NOCHANGE);

if (!prgListChanges)

{

\*pcChanges = m\_cChanges;

return S\_OK;

}

assert(\*pcChanges == 1);

prgListChanges[0] = m\_listChanges;

m\_fIsCounterDirty = FALSE;

m\_cChanges = 0;

m\_listChanges.Index = NULINDEX;

m\_listChanges.grfChange = TCT\_NOCHANGE;

return S\_OK;

}

private:

ULONG m\_uCounter;

ULONG m\_cChanges;

BOOL m\_fIsCounterDirty;

VSTREELISTITEMCHANGE m\_listChanges = { NULINDEX, TCT\_NOCHANGE };

}

Definition[] GetObjectLibraryDefinitions(wstring word)

{

Definition[] defs;

if(auto objmgr = queryService!(IVsObjectManager))

{

scope(exit) release(objmgr);

if(auto objmgr2 = qi\_cast!IVsObjectManager2(objmgr))

{

scope(exit) release(objmgr2);

IVsEnumLibraries2 enumLibs;

if(objmgr2.EnumLibraries(&enumLibs) == S\_OK)

{

VSOBSEARCHCRITERIA2 searchOpts;

searchOpts.szName = \_toUTF16zw(word);

searchOpts.eSrchType = SO\_ENTIREWORD;

searchOpts.grfOptions = VSOBSO\_CASESENSITIVE;

scope(exit) release(enumLibs);

DWORD fetched;

IVsLibrary2 lib;

while(enumLibs.Next(1, &lib, &fetched) == S\_OK && fetched == 1)

{

scope(exit) release(lib);

if(auto slib = qi\_cast!IVsSimpleLibrary2(lib))

{

scope(exit) release(slib);

IVsSimpleObjectList2 reslist;

if(slib.GetList2(LLT\_MEMBERS, LLF\_USESEARCHFILTER, &searchOpts, &reslist) == S\_OK)

{

scope(exit) release(reslist);

ULONG items;

if(reslist.GetItemCount(&items) == S\_OK && items > 0)

{

BOOL ok;

for(ULONG it = 0; it < items; it++)

if(reslist.CanGoToSource(it, GS\_DEFINITION, &ok) == S\_OK && ok)

{

}

}

}

}

}

}

}

}

return defs;

}

// This file is part of Visual D

//

// Visual D integrates the D programming language into Visual Studio

// Copyright (c) 2010 by Rainer Schuetze, All Rights Reserved

//

// Distributed under the Boost Software License, Version 1.0.

// See accompanying file LICENSE\_1\_0.txt or copy at <http://www.boost.org/LICENSE_1_0.txt>

module visuald.logutil;

import visuald.windows;

import std.format;

import std.utf;

import std.string;

import std.stdio;

import std.conv;

import std.datetime;

import std.array;

import stdcarg = core.stdc.stdarg;

import stdcio = core.stdc.stdio;

// import std.stdarg;

public import std.traits;

version(test) {} else {

import visuald.comutil;

public import visuald.vscommands;

static import dte = sdk.port.dte;

import sdk.win32.oleauto;

import sdk.vsi.textmgr;

import sdk.vsi.vsshell;

import sdk.vsi.vsshell80;

import sdk.vsi.vsshell90;

import sdk.vsi.ivssccmanager2;

import sdk.vsi.scguids;

import sdk.vsi.textmgr2;

import sdk.vsi.vssplash;

import sdk.vsi.fpstfmt;

import sdk.vsi.vsshlids;

import sdk.vsi.vsdebugguids;

import sdk.vsi.ocdesign;

import sdk.vsi.ivswebservices;

import sdk.vsi.encbuild;

///////////////////////////////////////////////////////////////

\_\_gshared bool \_false; // used in assert(false) to avoid semantic change of assert

///////////////////////////////////////////////////////////////

void OutputDebugLog(string msg)

{

OutputDebugStringA(toStringz(msg));

}

///////////////////////////////////////////////////////////////

T returnError(T)(T err)

{

logCall(" ERROR %x", err);

return err;

}

///////////////////////////////////////////////////////////////

const GUID IID\_IManagedObject = { 0xc3fcc19e, 0xa970, 0x11d2, [ 0x8b, 0x5a, 0x00, 0xa0, 0xc9, 0xb7, 0xc9, 0xc4 ] };

const GUID IID\_IRpcOptions = uuid("00000144-0000-0000-C000-000000000046");

const GUID IID\_SolutionProperties = uuid("28f7c3a6-fdc6-11d2-8a61-00c04f682e21");

const GUID IID\_isVCProject = uuid("3990034a-3af2-44c9-bd22-7b10654b5721");

const GUID IID\_GetActiveVCFileConfigurationFromVCFile1 = uuid("694c76bc-3ef4-11d3-b278-0050041db12a");

const GUID VisualD\_LanguageService = uuid("002a2de9-8bb6-484d-9800-7e4ad4084715");

string mixinGUID2string(string T)

{

return "static if (is(typeof(" ~ T ~ ") : GUID)) { if(guid == " ~ T ~ ") return \"" ~ T ~ "\"; }"

~ "else static if (is(typeof(" ~ T ~ ".iid) : GUID)) { if(guid == " ~ T ~ ".iid) return \"" ~ T ~ "\"; }"

~ "else static if (is(typeof(IID\_" ~ T ~ ") : GUID)) { if(guid == IID\_" ~ T ~ ") return \"" ~ T ~ "\"; }"

~ "else static if (is(typeof(uuid\_"~ T ~ ") : GUID)) { if(guid == uuid\_"~ T ~ ") return \"" ~ T ~ "\"; }"

~ "else static assert(0, \"unknown GUID " ~ T ~ "\");";

}

string GUID2utf8(ref const(GUID) guid)

{

mixin(mixinGUID2string("IUnknown"));

mixin(mixinGUID2string("IClassFactory"));

mixin(mixinGUID2string("IMarshal"));

mixin(mixinGUID2string("INoMarshal"));

mixin(mixinGUID2string("IAgileObject"));

mixin(mixinGUID2string("IMallocSpy"));

mixin(mixinGUID2string("IStdMarshalInfo"));

mixin(mixinGUID2string("IExternalConnection"));

mixin(mixinGUID2string("IMultiQI"));

mixin(mixinGUID2string("IEnumUnknown"));

mixin(mixinGUID2string("IBindCtx"));

mixin(mixinGUID2string("IEnumMoniker"));

mixin(mixinGUID2string("IRunnableObject"));

mixin(mixinGUID2string("IRunningObjectTable"));

mixin(mixinGUID2string("IPersist"));

mixin(mixinGUID2string("IPersistStream"));

mixin(mixinGUID2string("IMoniker"));

mixin(mixinGUID2string("IROTData"));

mixin(mixinGUID2string("IEnumString"));

mixin(mixinGUID2string("ISequentialStream"));

mixin(mixinGUID2string("IStream"));

mixin(mixinGUID2string("IEnumSTATSTG"));

mixin(mixinGUID2string("IStorage"));

mixin(mixinGUID2string("IPersistFile"));

mixin(mixinGUID2string("IPersistStorage"));

mixin(mixinGUID2string("ILockBytes"));

mixin(mixinGUID2string("IEnumFORMATETC"));

mixin(mixinGUID2string("IEnumSTATDATA"));

mixin(mixinGUID2string("IRootStorage"));

mixin(mixinGUID2string("IAdviseSink"));

mixin(mixinGUID2string("IAdviseSink2"));

mixin(mixinGUID2string("IDataObject"));

mixin(mixinGUID2string("IDataAdviseHolder"));

mixin(mixinGUID2string("IMessageFilter"));

mixin(mixinGUID2string("IRpcChannelBuffer"));

mixin(mixinGUID2string("IRpcProxyBuffer"));

mixin(mixinGUID2string("IRpcStubBuffer"));

mixin(mixinGUID2string("IPSFactoryBuffer"));

version(none)

{

//        mixin(mixinGUID2string("IPropertyStorage"));

//        mixin(mixinGUID2string("IPropertySetStorage"));

//        mixin(mixinGUID2string("IEnumSTATPROPSTG"));

//        mixin(mixinGUID2string("IEnumSTATPROPSETSTG"));

mixin(mixinGUID2string("IFillLockBytes"));

mixin(mixinGUID2string("IProgressNotify"));

mixin(mixinGUID2string("ILayoutStorage"));

//        mixin(mixinGUID2string("IRpcChannel"));

//        mixin(mixinGUID2string("IRpcStub"));

mixin(mixinGUID2string("IStubManager"));

mixin(mixinGUID2string("IRpcProxy"));

mixin(mixinGUID2string("IProxyManager"));

mixin(mixinGUID2string("IPSFactory"));

mixin(mixinGUID2string("IInternalMoniker"));

mixin(mixinGUID2string("IDfReserved1"));

mixin(mixinGUID2string("IDfReserved2"));

mixin(mixinGUID2string("IDfReserved3"));

mixin(mixinGUID2string("IStub"));

mixin(mixinGUID2string("IProxy"));

mixin(mixinGUID2string("IEnumGeneric"));

mixin(mixinGUID2string("IEnumHolder"));

mixin(mixinGUID2string("IEnumCallback"));

mixin(mixinGUID2string("IOleManager"));

mixin(mixinGUID2string("IOlePresObj"));

mixin(mixinGUID2string("IDebug"));

mixin(mixinGUID2string("IDebugStream"));

mixin(mixinGUID2string("StdOle"));

mixin(mixinGUID2string("ICreateTypeInfo"));

mixin(mixinGUID2string("ICreateTypeInfo2"));

mixin(mixinGUID2string("ICreateTypeLib"));

mixin(mixinGUID2string("ICreateTypeLib2"));

mixin(mixinGUID2string("IDispatch"));

mixin(mixinGUID2string("IEnumVARIANT"));

mixin(mixinGUID2string("ITypeComp"));

mixin(mixinGUID2string("ITypeInfo"));

mixin(mixinGUID2string("ITypeInfo2"));

mixin(mixinGUID2string("ITypeLib"));

mixin(mixinGUID2string("ITypeLib2"));

mixin(mixinGUID2string("ITypeChangeEvents"));

mixin(mixinGUID2string("IErrorInfo"));

mixin(mixinGUID2string("ICreateErrorInfo"));

mixin(mixinGUID2string("ISupportErrorInfo"));

mixin(mixinGUID2string("IOleAdviseHolder"));

mixin(mixinGUID2string("IOleCache"));

mixin(mixinGUID2string("IOleCache2"));

mixin(mixinGUID2string("IOleCacheControl"));

mixin(mixinGUID2string("IParseDisplayName"));

mixin(mixinGUID2string("IOleContainer"));

mixin(mixinGUID2string("IOleClientSite"));

mixin(mixinGUID2string("IOleObject"));

mixin(mixinGUID2string("IOleWindow"));

mixin(mixinGUID2string("IOleLink"));

mixin(mixinGUID2string("IOleItemContainer"));

mixin(mixinGUID2string("IOleInPlaceUIWindow"));

mixin(mixinGUID2string("IOleInPlaceActiveObject"));

mixin(mixinGUID2string("IOleInPlaceFrame"));

mixin(mixinGUID2string("IOleInPlaceObject"));

mixin(mixinGUID2string("IOleInPlaceSite"));

mixin(mixinGUID2string("IContinue"));

mixin(mixinGUID2string("IViewObject"));

mixin(mixinGUID2string("IViewObject2"));

mixin(mixinGUID2string("IEnumOLEVERB"));

}

mixin(mixinGUID2string("IDropSource"));

mixin(mixinGUID2string("IDropTarget"));

mixin(mixinGUID2string("IVsSccManager2"));

mixin(mixinGUID2string("IVsSccManager3"));

mixin(mixinGUID2string("IVsSccProject2"));

mixin(mixinGUID2string("IVsQueryEditQuerySave2"));

mixin(mixinGUID2string("IVsQueryEditQuerySave3"));

mixin(mixinGUID2string("IVsTrackProjectDocuments2"));

mixin(mixinGUID2string("IVsTrackProjectDocuments3"));

mixin(mixinGUID2string("IVsTrackProjectDocumentsEvents2"));

mixin(mixinGUID2string("IVsTrackProjectDocumentsEvents3"));

mixin(mixinGUID2string("IVsSccProviderFactory"));

mixin(mixinGUID2string("IVsSccProjectProviderBinding"));

mixin(mixinGUID2string("IVsSccProjectEnlistmentFactory"));

mixin(mixinGUID2string("IVsSccProjectEnlistmentChoice"));

mixin(mixinGUID2string("IVsSccEnlistmentPathTranslation"));

mixin(mixinGUID2string("IVsSccProjectFactoryUpgradeChoice"));

mixin(mixinGUID2string("SVsSccManager"));

mixin(mixinGUID2string("SVsQueryEditQuerySave"));

mixin(mixinGUID2string("SVsTrackProjectDocuments"));

mixin(mixinGUID2string("IVsLanguageInfo"));

mixin(mixinGUID2string("IVsLanguageDebugInfo"));

mixin(mixinGUID2string("IVsProvideColorableItems"));

mixin(mixinGUID2string("IVsColorableItem"));

mixin(mixinGUID2string("IVsLanguageContextProvider"));

mixin(mixinGUID2string("IVsLanguageBlock"));

mixin(mixinGUID2string("IServiceProvider"));

mixin(mixinGUID2string("IVsColorizer"));

mixin(mixinGUID2string("IVsColorizer2"));

mixin(mixinGUID2string("IVsDebuggerEvents"));

mixin(mixinGUID2string("IVsDebugger"));

mixin(mixinGUID2string("IVsFormatFilterProvider"));

mixin(mixinGUID2string("IVsCodeWindow"));

mixin(mixinGUID2string("IVsCodeWindowManager"));

mixin(mixinGUID2string("IVsTextBuffer"));

mixin(mixinGUID2string("IVsPackage"));

mixin(mixinGUID2string("IVsInstalledProduct"));

mixin(mixinGUID2string("IProfferService"));

mixin(mixinGUID2string("IVsTextLayer"));

mixin(mixinGUID2string("IVsLanguageTextOps"));

mixin(mixinGUID2string("IVsTextLines"));

mixin(mixinGUID2string("IVsTextView"));

mixin(mixinGUID2string("IVsEnumBSTR"));

mixin(mixinGUID2string("IVsUserDataEvents"));

mixin(mixinGUID2string("IVsTextLinesEvents"));

mixin(mixinGUID2string("IVsTextViewFilter"));

mixin(mixinGUID2string("IVsTextViewEvents"));

mixin(mixinGUID2string("IVsExpansionEvents"));

mixin(mixinGUID2string("IVsOutliningCapableLanguage"));

mixin(mixinGUID2string("IVsLanguageClipboardOps"));

mixin(mixinGUID2string("IVsProvideUserContextForObject"));

mixin(mixinGUID2string("IVsDynamicTabProvider"));

mixin(mixinGUID2string("IVsAutoOutliningClient"));

mixin(mixinGUID2string("IVsReadOnlyViewNotification"));

mixin(mixinGUID2string("IPreferPropertyPagesWithTreeControl"));

mixin(mixinGUID2string("IVsOutputWindowPane"));

mixin(mixinGUID2string("IVsProjectFactory"));

mixin(mixinGUID2string("IVsRegisterProjectTypes"));

mixin(mixinGUID2string("IVsHierarchy"));

mixin(mixinGUID2string("IVsUIHierarchy"));

mixin(mixinGUID2string("IVsOutput"));

mixin(mixinGUID2string("IVsEnumOutputs"));

mixin(mixinGUID2string("IVsCfg"));

mixin(mixinGUID2string("IVsProjectCfg"));

mixin(mixinGUID2string("IVsProjectCfg2"));

mixin(mixinGUID2string("IVsBuildableProjectCfg"));

mixin(mixinGUID2string("IVsBuildableProjectCfg2"));

mixin(mixinGUID2string("IVsBuildStatusCallback"));

mixin(mixinGUID2string("IVsDebuggableProjectCfg"));

mixin(mixinGUID2string("IVsCfgProvider"));

mixin(mixinGUID2string("IVsProjectCfgProvider"));

mixin(mixinGUID2string("IVsGetCfgProvider"));

mixin(mixinGUID2string("IVsProject"));

mixin(mixinGUID2string("IVsProject2"));

mixin(mixinGUID2string("IVsProject3"));

mixin(mixinGUID2string("IVsAggregatableProject"));

mixin(mixinGUID2string("IVsNonLocalProject"));

mixin(mixinGUID2string("IVsProjectFlavorCfg"));

mixin(mixinGUID2string("IPersist"));

mixin(mixinGUID2string("IPersistFileFormat"));

mixin(mixinGUID2string("IVsProjectBuildSystem"));

mixin(mixinGUID2string("IVsBuildPropertyStorage"));

mixin(mixinGUID2string("IVsComponentUser"));

mixin(mixinGUID2string("IVsDependencyProvider"));

mixin(mixinGUID2string("IVsDependency"));

mixin(mixinGUID2string("IVsEnumDependencies"));

mixin(mixinGUID2string("IVsProjectSpecialFiles"));

mixin(mixinGUID2string("IVsHierarchyEvents"));

mixin(mixinGUID2string("IVsPersistHierarchyItem"));

mixin(mixinGUID2string("IVsProjectSpecificEditorMap2"));

mixin(mixinGUID2string("IVsQueryLineChangeCommit"));

mixin(mixinGUID2string("IVsPersistDocData"));

mixin(mixinGUID2string("IVsCfgProvider2"));

mixin(mixinGUID2string("IVsParentProject"));

mixin(mixinGUID2string("IVsUpdateSolutionEvents"));

mixin(mixinGUID2string("IVsNonSolutionProjectFactory"));

mixin(mixinGUID2string("IVsProjectUpgradeViaFactory"));

mixin(mixinGUID2string("IVsProjectUpgrade"));

mixin(mixinGUID2string("IVsUpgradeLogger"));

mixin(mixinGUID2string("IVsProjectUpgradeViaFactory2"));

mixin(mixinGUID2string("IVsPersistSolutionOpts"));

mixin(mixinGUID2string("IVsSolutionPersistence"));

mixin(mixinGUID2string("IVsPersistSolutionProps"));

mixin(mixinGUID2string("IVsPublishableProjectCfg"));

mixin(mixinGUID2string("IVsPropertyPageNotify"));

mixin(mixinGUID2string("IVsPropertyPage"));

mixin(mixinGUID2string("IVsPropertyPage2"));

mixin(mixinGUID2string("IVsDeployableProjectCfg"));

mixin(mixinGUID2string("IConnectionPoint"));

mixin(mixinGUID2string("IManagedObject"));

mixin(mixinGUID2string("IProvideClassInfo"));

mixin(mixinGUID2string("IRpcOptions"));

mixin(mixinGUID2string("IEnumConnections"));

mixin(mixinGUID2string("IConnectionPointContainer"));

mixin(mixinGUID2string("IEnumConnectionPoints"));

mixin(mixinGUID2string("IOleCommandTarget"));

mixin(mixinGUID2string("IExtendedObject"));

mixin(mixinGUID2string("ISpecifyPropertyPages"));

mixin(mixinGUID2string("ISequentialStream"));

mixin(mixinGUID2string("IStream"));

mixin(mixinGUID2string("IPropertyBag"));

mixin(mixinGUID2string("IErrorLog"));

mixin(mixinGUID2string("IProvideMultipleClassInfo"));

mixin(mixinGUID2string("ISupportErrorInfo"));

mixin(mixinGUID2string("IUseImmediateCommitPropertyPages"));

mixin(mixinGUID2string("SolutionProperties"));

mixin(mixinGUID2string("isVCProject"));

mixin(mixinGUID2string("GetActiveVCFileConfigurationFromVCFile1"));

mixin(mixinGUID2string("dte.\_DTE"));

mixin(mixinGUID2string("dte.Project"));

mixin(mixinGUID2string("dte.Projects"));

mixin(mixinGUID2string("dte.ProjectItems"));

mixin(mixinGUID2string("dte.ProjectItem"));

mixin(mixinGUID2string("dte.Properties"));

mixin(mixinGUID2string("dte.Property"));

mixin(mixinGUID2string("CMDSETID\_StandardCommandSet2K"));

mixin(mixinGUID2string("CMDSETID\_StandardCommandSet97"));

mixin(mixinGUID2string("GUID\_VsUIHierarchyWindowCmds"));

mixin(mixinGUID2string("guidVSDebugCommand"));

//mixin(mixinGUID2string("VsSetGuidTeamSystemDataCmdIds"));

//mixin(mixinGUID2string("VsTextTransformationCmdIds"));

mixin(mixinGUID2string("IVsLanguageDebugInfoRemap"));

mixin(mixinGUID2string("IVsLanguageDebugInfo2"));

mixin(mixinGUID2string("IVsDebuggableProjectCfg2"));

mixin(mixinGUID2string("IVsENCRebuildableProjectCfg"));

mixin(mixinGUID2string("IVsWebServiceProvider"));

mixin(mixinGUID2string("VisualD\_LanguageService"));

return toUTF8(GUID2wstring(guid));

}

string tryformat(...)

{

string s;

void putc(dchar c)

{

s ~= c;

}

try {

std.format.doFormat(&putc, \_arguments, \_argptr);

}

catch(Exception e)

{

string msg = e.toString();

s ~= " EXCEPTION";

}

return s;

}

string \_tryformat(T)(T\* arg)

{

if(!arg)

return "null";

return tryformat("", \*arg);

}

string varToString(in VARIANT arg)

{

if (arg.vt == VT\_BSTR)

return to\_string(arg.bstrVal);

const VARIANT\_ALPHABOOL = 0x2;

int hr;

VARIANT temp;

hr = VariantChangeTypeEx(&temp, &arg, GetThreadLocale(), VARIANT\_ALPHABOOL, VT\_BSTR);

if (SUCCEEDED(hr))

return detachBSTR(temp.bstrVal);

return "invalid";

}

string \_toLog(GUID arg) { return GUID2utf8(arg); }

string \_toLog(in GUID\* arg) { return GUID2utf8(\*arg); }

string \_toLog(in VARIANT arg) { return format("VAR(%d,%s)", arg.vt, varToString(arg)); }

string \_toLog(string arg) { return arg; }

wstring \_toLog(in wchar\* arg) { return arg ? to\_wstring(arg) : "null"; }

void\* \_toLog(IUnknown arg) { return cast(void\*) arg; }

void\* \_toLog(in void\* arg) { return cast(void\*) arg; }

} // !version(test)

int \_toLog(int arg) { return arg; }

uint\* \_toLog(uint\* arg) { return arg; }

void\* \_toLog(Object arg) { return cast(void\*) arg; }

//T \_toLog(T)(T arg) { return arg; }

uint \_toLogOut(uint arg) { return arg; }

void\* \_toLogOut(IUnknown arg) { return cast(void\*) arg; }

string \_toLogOut(GUID arg) { return GUID2utf8(arg); }

version(all)

{

string \_toLogPtr(T)(const(T)\* arg)

{

static if(is(T : void)) return tryformat("", arg);

else static if(is(T : IUnknown)) return \_tryformat(cast(int\*\*)arg);

else static if(is(T : GUID)) return arg ? GUID2utf8(\*arg) : "null";

else static if(is(T : LARGE\_INTEGER)) return \_tryformat(cast(long\*)arg);

else static if(is(T : ULARGE\_INTEGER)) return \_tryformat(cast(ulong\*)arg);

else static if(is(T : IUnknown)) return arg ? \_tryformat(cast(int\*)\*arg) : "null";

else static if(is(T == struct)) return tryformat("struct ", cast(int\*)arg);

else return \_tryformat(arg);

}

} else { // !all

string \_toLogPtr(T : uint)(T\* arg) { return arg ? tryformat("%d", \*arg) : "null"; }

string \_toLogPtr(T : short)(T\* arg) { return arg ? tryformat("%s", arg) : "null"; }

string \_toLogPtr(T : wchar\*)(T\* arg) { return arg ? to\_string(\*arg) : "null"; }

string \_toLogPtr(T : void\*)(T\* arg) { return arg ? tryformat("", \*arg) : "null"; }

string \_toLogPtr(T : ulong)(T\* arg) { return arg ? tryformat("%d", \*arg) : "null"; }

string \_toLogPtr(T : IUnknown)(T\* arg) { return arg ? tryformat("", cast(int\*)\*arg) : "null"; }

version(test) {} else {

string \_toLogPtr(T : GUID)(T\* arg) { return GUID2utf8(\*arg); }

string \_toLogPtr(T : VARIANT)(T\* arg) { return arg ? \_toLog(\*arg) : "null"; }

string \_toLogPtr(T : LARGE\_INTEGER)(T\* arg) { return arg ? tryformat("%ld", arg.QuadPart) : "null"; }

string \_toLogPtr(T : ULARGE\_INTEGER)(T\* arg) { return arg ? tryformat("%ld", arg.QuadPart) : "null"; }

string \_toLogPtr(T : LPCOLESTR)(T arg) { return arg ? tryformat("%s", arg) : "null"; }

string \_toLogPtr(T : DISPPARAMS)(T\* arg) { return arg ? "struct" : "null"; }

string \_toLogPtr(T : EXCEPINFO)(T\* arg) { return arg ? "struct" : "null"; }

string \_toLogPtr(T : TYPEATTR)(T\* arg) { return arg ? "struct" : "null"; }

string \_toLogPtr(T : TYPEATTR\*)(T\* arg) { return arg ? tryformat("", \*arg) : "null"; }

string \_toLogPtr(T : FUNCDESC\*)(T\* arg) { return arg ? tryformat("", \*arg) : "null"; }

string \_toLogPtr(T : VARDESC\*)(T\* arg) { return arg ? tryformat("", \*arg) : "null"; }

string \_toLogPtr(T : PVSCOMPONENTSELECTORDATA)(T\* arg) { return arg ? tryformat("", \*arg) : "null"; }

string \_toLogPtr(T : CALPOLESTR\*)(T\* arg) { return arg ? tryformat("", \*arg) : "null"; }

string \_toLogPtr(T : FUNCDESC)(T\* arg) { return "in"; }

string \_toLogPtr(T : VARDESC)(T\* arg) { return "in"; }

string \_toLogPtr(T : CAUUID)(T\* arg) { return "cauuid"; }

string \_toLogPtr(T : CALPOLESTR)(T\* arg) { return arg ? tryformat("", arg) : "null"; }

string \_toLogPtr(T : CADWORD)(T\* arg) { return arg ? tryformat("", arg) : "null"; }

string \_toLogPtr(T : OLECMD)(T\* arg) { return arg ? tryformat("", arg) : "null"; }

string \_toLogPtr(T : OLECMDTEXT)(T\* arg) { return arg ? tryformat("", arg) : "null"; }

string \_toLogPtr(T : RECT)(T\* arg) { return arg ? tryformat("", arg) : "null"; }

string \_toLogPtr(T : TextSpan)(T\* arg) { return arg ? tryformat("", arg) : "null"; }

} // !version(test)

} // !all

int gLogIndent = 0;

\_\_gshared bool gLogFirst = true;

const string gLogFile = "c:/tmp/visuald.log";

const string gLogGCFile = "c:/tmp/visuald.gc";

void logIndent(int n)

{

gLogIndent += n;

}

\_\_gshared FILE\* gcLogFh;

extern(C) void log\_printf(string fmt, ...)

{

stdcarg.va\_list q;

stdcarg.va\_start!(string)(q, fmt);

char[256] buf;

int len = vsprintf(buf.ptr, fmt.ptr, q);

if(!gcLogFh)

gcLogFh = stdcio.fopen(gLogGCFile.ptr, "w");

if(gcLogFh)

stdcio.fwrite(buf.ptr, len, 1, gcLogFh);

stdcarg.va\_end(q);

}

extern(C) void log\_flush()

{

if(gcLogFh)

stdcio.fflush(gcLogFh);

}

version(test) {

void logCall(...)

{

string s;

void putc(dchar c)

{

s ~= c;

}

std.format.doFormat(&putc, \_arguments, \_argptr);

s ~= "\n";

std.stdio.fputs(toStringz(s), stdout.getFP);

}

} else debug {

class logSync {}

void logCall(...)

{

auto buffer = new char[32];

SysTime now = Clock.currTime();

uint tid = GetCurrentThreadId();

auto len = sprintf(buffer.ptr, "%02d:%02d:%02d - %04x - ",

now.hour, now.minute, now.second, tid);

string s = to!string(buffer[0..len]);

s ~= replicate(" ", gLogIndent);

void putc(dchar c)

{

s ~= c;

}

try {

std.format.doFormat(&putc, \_arguments, \_argptr);

}

catch(Exception e)

{

string msg = e.toString();

s ~= " EXCEPTION";

}

log\_string(s);

}

void log\_string(string s)

{

s ~= "\n";

if(gLogFile.length == 0)

OutputDebugStringA(toStringz(s));

else

synchronized(logSync.classinfo)

{

static \_\_gshared bool canLog;

if(gLogFirst)

{

gLogFirst = false;

s = "\n" ~ replicate("=", 80) ~ "\n" ~ s;

try

{

string bar = "\n" ~ replicate("=", 80) ~ "\n";

std.file.write(gLogFile, bar); // append?

canLog = true;

}

catch(Exception e)

{

}

}

if(canLog)

{

try

{

std.file.append(gLogFile, s);

}

catch(Exception e)

{

}

}

}

}

}

else

{

void logCall(...)

{

}

void log\_string(string s)

{

}

}

/////////////////////////////////////////////////////////////////////

// Parsing mangles for fun and profit.

string \_getJustName(string mangle)

{

size\_t idx = 1;

size\_t start = idx;

size\_t len = 0;

while(idx < mangle.length && mangle[idx] >= '0' &&

mangle[idx] <= '9')

{

int size = mangle[idx++] - '0';

while(mangle[idx] >= '0' && mangle[idx] <= '9')

size = (size \* 10) + (mangle[idx++] - '0');

start = idx;

len = size;

idx += size;

}

if(start < mangle.length)

return mangle[start .. start + len];

else

return "";

}

// get anything between first '(' and last ')'

string \_getArgs(string func)

{

int sidx = 0;

if(startsWith(func, "extern ("))

sidx += 8;

while(sidx < func.length && func[sidx] != '(')

sidx++;

int eidx = func.length - 1;

while(eidx >= 0 && func[eidx] != ')')

eidx++;

if(sidx < eidx)

return func[sidx + 1 .. eidx];

return "";

}

string \_nextArg(string args)

{

int sidx = 0;

while(sidx < args.length && args[sidx] != ',')

sidx++;

if(sidx < args.length)

return args[sidx + 1 .. args.length];

return "";

}

int \_find(string s, char c)

{

for(int i = 0; i < s.length; i++)

if(s[i] == c)

return i;

return -1;

}

string \_getIdentifier(string args)

{

string ident;

int sidx = -1;

for(int idx = 0; ; idx++)

{

dchar ch = (idx < args.length ? args[idx] : ',');

if(sidx >= 0)

{

if(!((ch >= 'A' && ch <= 'Z') || (ch >= 'a' && ch <= 'z') ||

(ch >= '0' && ch <= '9') || ch == '\_'))

{

ident = args[sidx .. idx];

sidx = -1;

}

}

else if((ch >= 'A' && ch <= 'Z') || (ch >= 'a' && ch <= 'z') || ch == '\_')

sidx = idx;

if(ch == ',')

break;

}

return ident;

}

string \_toArgIdx(int idx)

{

string s = "";

if(idx == 0)

s = "0";

else

while(idx > 0)

{

s = cast(char)('0' + (idx % 10)) ~ s;

idx = idx / 10;

}

return s;

}

// useargs: 0 - identifier, 1 - C-style, 2 - D-style

string \_getLogCall(string func, string type, bool addthis, int useargs)

{

string call = "logCall(\"";

string args = \_getArgs(type);

string idlist;

if(addthis)

{

call ~= "%s.";

idlist ~= ", this";

}

call ~= func ~ "(";

int arg = 0;

if(addthis)

{

call ~= "this=%s";

idlist ~= ", cast(void\*)this";

arg = 1;

}

while(args.length > 0)

{

bool isOut = (args.length > 4 && args[0 .. 4] == "out ");

string ident = \_getIdentifier(args);

if(ident.length > 0)

{

if(arg > 0)

call ~= ", ";

if(useargs == 0)

{

call ~= (isOut ? "out " : "") ~ ident ~ "=%" ~ 's'; // cast(char)(arg + '1');

idlist ~= ", \_toLog(" ~ (isOut ? "&" : "") ~ ident ~ ")";

}

else

{

call ~= "%" ~ 's'; // cast(char)(arg + '1');

string sidx = \_toArgIdx(addthis ? arg - 1 : arg);

idlist ~= ", \_toLog(\*cast(\_argtypes[" ~ sidx ~ "]\*)(\_ebp+\_argoff(" ~ \_toArgIdx(arg) ~ ")))";

}

arg++;

}

args = \_nextArg(args);

}

call ~= ")\"" ~ idlist ~ ");\n";

return call;

}

string \_getLogReturn(string func, string type)

{

string call = "logCall(\"" ~ func ~ " returns ";

string args = \_getArgs(type);

string idlist;

int arg = 0;

while(args.length > 0)

{

string prevargs = args;

bool isOut = (args.length > 4 && args[0 .. 4] == "out ");

string ident = \_getIdentifier(args);

args = \_nextArg(args);

bool isPtr = false;

if(!isOut)

{

int len = prevargs.length;

if((len < 5 || prevargs[0..5] != "void\*") &&

(len < 4 || prevargs[0..4] != "MSG\*") &&

(len < 13 || prevargs[0..13] != "PROPPAGEINFO\*"))

{

int idx = \_find(prevargs, '\*');

isPtr = (idx >= 0 && idx < prevargs.length - args.length);

}

}

if(ident.length > 0 && (isOut || isPtr))

{

if(arg > 0)

call ~= ", ";

call ~= ident ~ "=%s";

if(isOut)

idlist ~= ", \_toLogOut(" ~ ident ~ ")";

else

idlist ~= ", \_toLogPtr(" ~ ident ~ ")";

arg++;

}

}

if(arg == 0)

return "";

call ~= "\"" ~ idlist ~ ");\n";

return call;

}

///////////////////////////////////////////////

const string nl = " "; // "\n";

debug {

version(all)

{

string genLogMixin(PI...)(string fn, bool hasThis)

{

string fmt;

string args;

if(hasThis)

{

fmt = "this=%s";

args = ", cast(void\*) this";

}

foreach(id; PI)

{

args ~= ", \_toLog(" ~ id ~ ")";

if(fmt.length > 0)

fmt ~= ", ";

fmt ~= id ~ "=%s";

}

return "logCall(\"" ~ fn ~ "(" ~ fmt ~ ")\"" ~ args ~ ");";

}

template logParameterIdentifier(alias sym)

{

import std.typetuple;

import std.traits;

alias TypeTuple!(\_\_traits(parent,sym)) func;

alias ParameterIdentifierTuple!(func[0]) logParameterIdentifier;

//alias ParameterTypeTuple!(func[0]) PT;

}

const string \_logMix = q{

enum LogCall = genLogMixin!(logParameterIdentifier!hasThis)(\_\_FUNCTION\_\_, hasThis);

mixin(LogCall);

};

const string \_hasThisMix = "const bool hasThis = true;" ~ nl;

const string \_LogIndentNoRet = "logIndent(1); scope(exit) logIndent(-1);" ~ nl;

const string LogCallMix = \_hasThisMix ~ \_logMix ~ \_LogIndentNoRet;

const string LogCallMix2 = \_hasThisMix ~ \_logMix ~ \_LogIndentNoRet;

const string LogCallMixNoRet = \_hasThisMix ~ \_logMix ~ \_LogIndentNoRet;

}

else

{

const string FuncNameMix = "struct \_\_FUNCTION {} static const string \_FUNCTION\_ = \_getJustName(\_\_FUNCTION.mangleof);" ~ nl;

const string \_hasThisMix = "static const bool hasThis = true;" ~ nl;

const string \_LogCallMix = "static const string \_\_LOGCALL\_\_ = \_getLogCall(\_FUNCTION\_, typeof(&mixin(\_FUNCTION\_)).stringof, hasThis, 0);" ~ nl;

const string \_LogReturnMix = "static const string \_\_LOGRETURN\_\_ = \_getLogReturn(\_FUNCTION\_, typeof(&mixin(\_FUNCTION\_)).stringof);" ~ nl;

const string \_getEBP = "byte\* \_ebp; asm { mov \_ebp,EBP; } \_ebp = \_ebp + 8;" ~ nl;

const string \_LogCallArgType = "static const string \_\_ARGTYPES\_\_ = \"alias ParameterTypeTuple!(\" ~ \_FUNCTION\_ ~ \") \_argtypes;\";" ~ nl;

const string \_LogCallArgOff = "static int \_argoff(int n) { int off = 0; foreach(i, T; \_argtypes) if(i < n) off += T.sizeof; return off; }" ~ nl;

const string \_LogCallMix2 = "static const string \_\_LOGCALL\_\_ = \_getLogCall(\_FUNCTION\_, typeof(&mixin(\_FUNCTION\_)).stringof, hasThis, 1);" ~ nl;

const string \_LogIndent = "logIndent(1); scope(exit) { " ~ "mixin(\_\_LOGRETURN\_\_);" ~ "logIndent(-1); }" ~ nl;

const string \_LogIndentNoRet = "logIndent(1); scope(exit) logIndent(-1);" ~ nl;

const string LogCallMix = FuncNameMix ~ \_hasThisMix ~ \_LogCallMix ~ \_LogReturnMix ~ "mixin(\_\_LOGCALL\_\_);" ~ \_LogIndent;

const string LogCallMix2 = FuncNameMix ~ \_hasThisMix ~ \_getEBP ~ \_LogCallArgType ~ "mixin(\_\_ARGTYPES\_\_);" ~ \_LogCallArgOff ~ \_LogCallMix2 ~ "mixin(\_\_LOGCALL\_\_);" ~ \_LogIndentNoRet;

const string LogCallMixNoRet = FuncNameMix ~ \_hasThisMix ~ \_LogCallMix ~ "mixin(\_\_LOGCALL\_\_);" ~ \_LogIndentNoRet;

}

} else {

const string LogCallMix = "";

const string LogCallMix2 = "";

const string LogCallMixNoRet = "";

}

/+

void test(int a0, Object o)

{

mixin(FuncNameMix);

pragma(msg, \_FUNCTION\_); // shows "test"

pragma(msg,typeof(&mixin(\_FUNCTION\_)).stringof); // shows "void function(int a0, Object o)"

pragma(msg,\_getLogCall(\_FUNCTION\_, typeof(&mixin(\_FUNCTION\_)).stringof, false)); // shows "void function(int a0, Object o)"

}

+/

/+

template tLogCall(alias s)

{

struct \_\_STRUCT {};

static const string \_FUNCTION\_ = \_getJustName(\_\_STRUCT.mangleof);

//pragma(msg, s.mangleof); // shows "test"

pragma(msg, \_\_STRUCT.mangleof); // shows "test"

pragma(msg, \_FUNCTION\_); // shows "test"

alias ParameterTypeTuple!(test2) types;

void\* pthis = cast(void\*)this;

}

class t

{

void test2(int a0, Object o, uint x)

{

struct \_\_STR {}

mixin tLogCall!(\_\_STR);

}

}

+/

version(test) {

import std.stdio;

import std.string;

template log\_arg(T)

{

T log\_arg(inout void\* \_argptr)

{

T arg = \*cast(T\*)\_argptr;

\_argptr = \_argptr + ((T.sizeof + int.sizeof - 1) & ~(int.sizeof - 1));

return arg;

}

}

class t

{

void test2(int a0, Object o, uint x)

{

mixin(LogCallMix);

alias ParameterTypeTuple!(test2) types;

pragma(msg,types.stringof);

TypeInfo[] ti;

foreach\_reverse(t; types)

{

pragma(msg,t.stringof);

ti ~= typeid(t);

}

void\* pthis = cast(void\*)this;

void\* p; asm { mov p,EBP; } p = p + 8;

std.format.doFormat(&putc, ti, p);

logCall("doFormat = %s", s);

auto arg3 = log\_arg!(types[2])(p);

auto arg2 = log\_arg!(types[1])(p);

auto arg1 = log\_arg!(types[0])(p);

logCall("%s.test2(this=%s,a0=%s,o=%s,x=%s)", this, pthis, \_toLog(arg1), \_toLog(arg2), \_toLog(arg3));

int \*vp = cast(int\*) &this;

for(int i = -6; i < 6; i++)

logCall("%d: %x", i, vp[i]);

}

}

int rc = 2;

int main(char[][] argv)

{

t at = new t;

at.test2(3, null, 7);

return rc;

}

}

// This file is part of Visual D

//

// Visual D integrates the D programming language into Visual Studio

// Copyright (c) 2010 by Rainer Schuetze, All Rights Reserved

//

// Distributed under the Boost Software License, Version 1.0.

// See accompanying file LICENSE\_1\_0.txt or copy at <http://www.boost.org/LICENSE_1_0.txt>

module visuald.oledatasource;

import core.stdc.string : memcmp, memset, memcpy;

import visuald.windows;

import sdk.win32.objbase;

import sdk.win32.objidl;

import visuald.comutil;

import visuald.hierutil;

import visuald.logutil;

extern(Windows)

{

void ReleaseStgMedium(in STGMEDIUM\* medium);

}

struct VX\_DATACACHE\_ENTRY

{

FORMATETC m\_formatEtc;

STGMEDIUM m\_stgMedium;

DATADIR m\_nDataDir;

};

//---------------------------------------------------------------------------

//---------------------------------------------------------------------------

class OleDataSource : DComObject, IDataObject

{

VX\_DATACACHE\_ENTRY[] mCache;

IDataAdviseHolder mDataAdviseHolder;

~this()

{

// free the clipboard data cache

Empty();

}

override HRESULT QueryInterface(in IID\* riid, void\*\* pvObject)

{

//mixin(LogCallMix);

if(queryInterface!(IDataObject) (this, riid, pvObject))

return S\_OK;

return super.QueryInterface(riid, pvObject);

}

//---------------------------------------------------------------------------

void Empty()

{

// release all of the STGMEDIUMs and FORMATETCs

for (UINT nIndex = 0; nIndex < mCache.length; nIndex++)

{

CoTaskMemFree(mCache[nIndex].m\_formatEtc.ptd);

.ReleaseStgMedium(&mCache[nIndex].m\_stgMedium);

}

mCache.length = 0;

mDataAdviseHolder = release(mDataAdviseHolder);

}

/+

/////////////////////////////////////////////////////////////////////////////

// OleDataSource clipboard API wrappers

void SetClipboard(void)

{

// attempt OLE set clipboard operation

SCODE sc = ::OleSetClipboard(this);

ASSERT(S\_OK == sc);

sc;

// success - set as current clipboard source

// \_afxOleState.m\_pClipboardSource = this;

ASSERT(::OleIsCurrentClipboard(this) == S\_OK);

}

void PASCAL OleDataSource::FlushClipboard()

{

if (GetClipboardOwner() != null)

{

// active clipboard source and it is on the clipboard - flush it

::OleFlushClipboard();

// shouldn't be clipboard owner any more...

ASSERT(GetClipboardOwner() == null);

}

}

#if 0

OleDataSource\* PASCAL OleDataSource::GetClipboardOwner()

{

\_AFX\_OLE\_STATE\* pOleState = \_afxOleState;

if (pOleState.m\_pClipboardSource == null)

return null; // can't own the clipboard if pClipboardSource isn't set

ASSERT\_VALID(pOleState.m\_pClipboardSource);

LPDATAOBJECT lpDataObject = (LPDATAOBJECT)

pOleState.m\_pClipboardSource.GetInterface(&IID\_IDataObject);

if (::OleIsCurrentClipboard(lpDataObject) != S\_OK)

{

pOleState.m\_pClipboardSource = null;

return null; // don't own the clipboard anymore

}

// return current clipboard sourcew

return pOleState.m\_pClipboardSource;

}

#endif

+/

/////////////////////////////////////////////////////////////////////////////

// OleDataSource cache allocation

VX\_DATACACHE\_ENTRY\* GetCacheEntry(FORMATETC\* lpFormatEtc, DATADIR nDataDir)

{

VX\_DATACACHE\_ENTRY\* pEntry = Lookup(lpFormatEtc, nDataDir);

if (pEntry)

{

// cleanup current entry and return it

CoTaskMemFree(pEntry.m\_formatEtc.ptd);

.ReleaseStgMedium(&pEntry.m\_stgMedium);

}

else

{

// allocate space for item at m\_nSize (at least room for 1 item)

mCache.length = mCache.length + 1;

pEntry = &mCache[$-1];

}

// fill the cache entry with the format and data direction and return it

pEntry.m\_nDataDir = nDataDir;

pEntry.m\_formatEtc = \*lpFormatEtc;

return pEntry;

}

/////////////////////////////////////////////////////////////////////////////

// OleDataSource operations

// for HGLOBAL based cached render

void CacheGlobalData(CLIPFORMAT cfFormat, HGLOBAL hGlobal, FORMATETC\* lpFormatEtc)

{

// fill in FORMATETC struct

FORMATETC formatEtc;

lpFormatEtc = \_FillFormatEtc(lpFormatEtc, cfFormat, &formatEtc);

assert(lpFormatEtc);

if(!lpFormatEtc)

return;

lpFormatEtc.tymed = TYMED\_HGLOBAL;

// add it to the cache

VX\_DATACACHE\_ENTRY\* pEntry = GetCacheEntry(lpFormatEtc, DATADIR\_GET);

pEntry.m\_stgMedium.tymed = TYMED\_HGLOBAL;

pEntry.m\_stgMedium.hGlobal = hGlobal;

pEntry.m\_stgMedium.pUnkForRelease = null;

}

// for raw STGMEDIUM\* cached render

void CacheData(CLIPFORMAT cfFormat, STGMEDIUM\* lpStgMedium, FORMATETC\* lpFormatEtc)

{

// fill in FORMATETC struct

FORMATETC formatEtc;

lpFormatEtc = \_FillFormatEtc(lpFormatEtc, cfFormat, &formatEtc);

// Only these TYMED\_GDI formats can be copied, so can't serve as

// cache content (you must use DelayRenderData instead)

// When using COleServerItem::CopyToClipboard this means providing an

// override of COleServerItem::OnGetClipboardData to provide a custom

// delayed rendering clipboard object.

assert(lpStgMedium.tymed != TYMED\_GDI ||

lpFormatEtc.cfFormat == CF\_METAFILEPICT ||

lpFormatEtc.cfFormat == CF\_PALETTE ||

lpFormatEtc.cfFormat == CF\_BITMAP);

lpFormatEtc.tymed = lpStgMedium.tymed;

// add it to the cache

VX\_DATACACHE\_ENTRY\* pEntry = GetCacheEntry(lpFormatEtc, DATADIR\_GET);

pEntry.m\_stgMedium = \*lpStgMedium;

}

// for STGMEDIUM\* or HGLOBAL based delayed render

void DelayRenderData(CLIPFORMAT cfFormat, FORMATETC\* lpFormatEtc)

{

// fill in FORMATETC struct

FORMATETC formatEtc;

if (lpFormatEtc is null)

{

lpFormatEtc = \_FillFormatEtc(lpFormatEtc, cfFormat, &formatEtc);

lpFormatEtc.tymed = TYMED\_HGLOBAL;

}

// insure that cfFormat member is set

if (cfFormat != 0)

lpFormatEtc.cfFormat = cfFormat;

// add it to the cache

VX\_DATACACHE\_ENTRY\* pEntry = GetCacheEntry(lpFormatEtc, DATADIR\_GET);

pEntry.m\_stgMedium = pEntry.m\_stgMedium;

}

//---------------------------------------------------------------------------

// DelaySetData -- used to allow SetData on given FORMATETC\*

//---------------------------------------------------------------------------

void DelaySetData(CLIPFORMAT cfFormat, FORMATETC\* lpFormatEtc)

{

// fill in FORMATETC struct

FORMATETC formatEtc;

lpFormatEtc = \_FillFormatEtc(lpFormatEtc, cfFormat, &formatEtc);

// add it to the cache

VX\_DATACACHE\_ENTRY\* pEntry = GetCacheEntry(lpFormatEtc, DATADIR\_SET);

pEntry.m\_stgMedium.tymed = TYMED\_NULL;

pEntry.m\_stgMedium.hGlobal = null;

pEntry.m\_stgMedium.pUnkForRelease = null;

}

/////////////////////////////////////////////////////////////////////////////

// OleDataSource cache implementation

VX\_DATACACHE\_ENTRY\* Lookup(in FORMATETC\* lpFormatEtc, DATADIR nDataDir)

{

VX\_DATACACHE\_ENTRY\* pLast = null;

// look for suitable match to lpFormatEtc in cache

for (UINT nIndex = 0; nIndex < mCache.length; nIndex++)

{

// get entry from cache at nIndex

VX\_DATACACHE\_ENTRY\* pCache = &mCache[nIndex];

FORMATETC \*pCacheFormat = &pCache.m\_formatEtc;

// check for match

if (pCacheFormat.cfFormat == lpFormatEtc.cfFormat &&

(pCacheFormat.tymed & lpFormatEtc.tymed) != 0 &&

pCacheFormat.lindex == lpFormatEtc.lindex &&

pCacheFormat.dwAspect == lpFormatEtc.dwAspect &&

pCache.m\_nDataDir == nDataDir)

{

// for backward compatibility we match even if we never

// find an exact match for the DVTARGETDEVICE

const(DVTARGETDEVICE)\* ptd1 = pCacheFormat.ptd;

const(DVTARGETDEVICE)\* ptd2 = lpFormatEtc.ptd;

pLast = pCache;

if(((ptd1 is null) && (ptd2 is null)) ||

((ptd1 !is null) && (ptd2 !is null) &&

(ptd1.tdSize == ptd2.tdSize) &&

(memcmp(ptd1, ptd2, ptd1.tdSize)==0)

))

{

// exact match, so break now and return it

break;

}

// continue looking for better match

}

}

return pLast;

}

/////////////////////////////////////////////////////////////////////////////

// OleDataSource overidable default implementation

BOOL OnRenderGlobalData(in FORMATETC\* lpFormatEtc, HGLOBAL\* phGlobal)

{

return FALSE; // default does nothing

}

/+

//---------------------------------------------------------------------------

BOOL OnRenderFileData(FORMATETC\* lpFormatEtc, CVsFile\* /\*pFile\*/)

{

return FALSE; // default does nothing

}

+/

//---------------------------------------------------------------------------

BOOL OnRenderData(in FORMATETC\* lpFormatEtc, STGMEDIUM\* lpStgMedium)

{

// attempt TYMED\_HGLOBAL as prefered format

if (lpFormatEtc.tymed & TYMED\_HGLOBAL)

{

// attempt HGLOBAL delay render hook

HGLOBAL hGlobal = lpStgMedium.hGlobal;

if (OnRenderGlobalData(lpFormatEtc, &hGlobal))

{

assert(lpStgMedium.tymed != TYMED\_HGLOBAL || (lpStgMedium.hGlobal == hGlobal));

assert(hGlobal != null);

lpStgMedium.tymed = TYMED\_HGLOBAL;

lpStgMedium.hGlobal = hGlobal;

return TRUE;

}

/+

// attempt CVsFile\* based delay render hook

CVsSharedFile file;

if (lpStgMedium.tymed == TYMED\_HGLOBAL)

{

ASSERT(lpStgMedium.hGlobal != null);

file.SetHandle(lpStgMedium.hGlobal, FALSE);

}

if (OnRenderFileData(lpFormatEtc, &file))

{

lpStgMedium.tymed = TYMED\_HGLOBAL;

lpStgMedium.hGlobal = file.Detach();

ASSERT(lpStgMedium.hGlobal != null);

return TRUE;

}

if (lpStgMedium.tymed == TYMED\_HGLOBAL)

file.Detach();

+/

}

/+

// attempt TYMED\_ISTREAM format

if (lpFormatEtc.tymed & TYMED\_ISTREAM)

{

ASSERT(!\_T("port COleStreamFile"));

#if 0

COleStreamFile file;

if (lpStgMedium.tymed == TYMED\_ISTREAM)

{

ASSERT(lpStgMedium.pstm != null);

file.Attach(lpStgMedium.pstm);

}

else

{

if (!file.CreateMemoryStream())

return FALSE;

}

// get data into the stream

if (OnRenderFileData(lpFormatEtc, &file))

{

lpStgMedium.tymed = TYMED\_ISTREAM;

lpStgMedium.pstm = file.Detach();

return TRUE;

}

if (lpStgMedium.tymed == TYMED\_ISTREAM)

file.Detach();

#endif //0

}

+/

return FALSE; // default does nothing

}

//---------------------------------------------------------------------------

BOOL OnSetData(in FORMATETC\* lpFormatEtc, in STGMEDIUM\* lpStgMedium, BOOL bRelease)

{

return FALSE; // default does nothing

}

//---------------------------------------------------------------------------

override HRESULT GetData(/\* [unique][in] \*/ in FORMATETC \*pformatetcIn,

/\* [out] \*/ STGMEDIUM \*pmedium)

{

mixin(LogCallMix2);

// attempt to find match in the cache

VX\_DATACACHE\_ENTRY\* pCache = Lookup(pformatetcIn, DATADIR\_GET);

if (!pCache)

return DV\_E\_FORMATETC;

// use cache if entry is not delay render

memset(pmedium, 0, STGMEDIUM.sizeof);

if (pCache.m\_stgMedium.tymed != TYMED\_NULL)

{

// Copy the cached medium into the lpStgMedium provided by caller.

if (!\_CopyStgMedium(pformatetcIn.cfFormat, pmedium, &pCache.m\_stgMedium))

return DV\_E\_FORMATETC;

// format was supported for copying

return S\_OK;

}

SCODE sc = DV\_E\_FORMATETC;

// attempt STGMEDIUM\* based delay render

if (OnRenderData(pformatetcIn, pmedium))

sc = S\_OK;

return sc;

}

//---------------------------------------------------------------------------

override HRESULT GetDataHere(/\* [unique][in] \*/ in FORMATETC \*pformatetc,

/\* [out][in] \*/ STGMEDIUM \*pmedium)

{

mixin(LogCallMix2);

// these two must be the same

assert(pformatetc.tymed == pmedium.tymed);

// pformatetc.tymed = pmedium.tymed; // but just in case...

// attempt to find match in the cache

VX\_DATACACHE\_ENTRY\* pCache = Lookup(pformatetc, DATADIR\_GET);

if (!pCache)

return DV\_E\_FORMATETC;

// handle cached medium and copy

if (pCache.m\_stgMedium.tymed != TYMED\_NULL)

{

// found a cached format -- copy it to dest medium

assert(pCache.m\_stgMedium.tymed == pmedium.tymed);

if (!\_CopyStgMedium(pformatetc.cfFormat, pmedium, &pCache.m\_stgMedium))

return DV\_E\_FORMATETC;

// format was supported for copying

return S\_OK;

}

SCODE sc = DV\_E\_FORMATETC;

// attempt pmedium based delay render

if (OnRenderData(pformatetc, pmedium))

sc = S\_OK;

return sc;

}

//---------------------------------------------------------------------------

override HRESULT QueryGetData(/\* [unique][in] \*/ in FORMATETC \*pformatetc)

{

mixin(LogCallMix2);

// attempt to find match in the cache

VX\_DATACACHE\_ENTRY\* pCache = Lookup(pformatetc, DATADIR\_GET);

if (!pCache)

return DV\_E\_FORMATETC;

// it was found in the cache or can be rendered -- success

return S\_OK;

}

//---------------------------------------------------------------------------

override HRESULT GetCanonicalFormatEtc(/\* [unique][in] \*/ in FORMATETC \*pformatectIn,

/\* [out] \*/ FORMATETC \*pformatetcOut)

{

mixin(LogCallMix2);

// because we support the target-device (ptd) for server metafile format,

// all members of the FORMATETC are significant.

return DATA\_S\_SAMEFORMATETC;

}

//---------------------------------------------------------------------------

override HRESULT SetData(/\* [unique][in] \*/ in FORMATETC \*pformatetc,

/\* [unique][in] \*/ in STGMEDIUM \*pmedium,

/\* [in] \*/ in BOOL fRelease)

{

mixin(LogCallMix2);

assert(pformatetc.tymed == pmedium.tymed);

// attempt to find match in the cache

VX\_DATACACHE\_ENTRY\* pCache = Lookup(pformatetc, DATADIR\_SET);

if (!pCache)

return DV\_E\_FORMATETC;

assert(pCache.m\_stgMedium.tymed == TYMED\_NULL);

SCODE sc = E\_UNEXPECTED;

// attempt pmedium based SetData

if (OnSetData(pformatetc, pmedium, fRelease))

sc = S\_OK;

return sc;

}

//---------------------------------------------------------------------------

override HRESULT EnumFormatEtc(/\* [in] \*/ in DWORD dwDirection,

/\* [out] \*/ IEnumFORMATETC \*ppenumFormatEtc)

{

mixin(LogCallMix2);

\*ppenumFormatEtc = null;

// generate a format list from the cache

CEnumFormatEtc pFormatList = newCom!CEnumFormatEtc(this, dwDirection);

\*ppenumFormatEtc = addref(pFormatList);

return S\_OK;

}

//---------------------------------------------------------------------------

override HRESULT DAdvise(/\* [in] \*/ in FORMATETC \*pformatetc,

/\* [in] \*/ in DWORD advf,

/\* [unique][in] \*/ IAdviseSink pAdvSink,

/\* [out] \*/ DWORD \*pdwConnection)

{

mixin(LogCallMix2);

HRESULT hr = S\_OK;

if (!mDataAdviseHolder)

hr = CreateDataAdviseHolder(&mDataAdviseHolder);

if (hr == S\_OK)

hr = mDataAdviseHolder.Advise(this, pformatetc, advf, pAdvSink, pdwConnection);

return hr;

}

//---------------------------------------------------------------------------

override HRESULT DUnadvise(/\* [in] \*/ in DWORD dwConnection)

{

mixin(LogCallMix2);

HRESULT hr = OLE\_E\_NOCONNECTION;

if (mDataAdviseHolder)

hr = mDataAdviseHolder.Unadvise(dwConnection);

return hr;

}

//---------------------------------------------------------------------------

override HRESULT EnumDAdvise(/\* [out] \*/ IEnumSTATDATA \*ppenumAdvise)

{

mixin(LogCallMix2);

HRESULT hr = E\_FAIL;

if (mDataAdviseHolder)

hr = mDataAdviseHolder.EnumAdvise(ppenumAdvise);

return hr;

}

}

//---------------------------------------------------------------------------

class CEnumFormatEtc : DComObject, IEnumFORMATETC

{

this(OleDataSource src, DWORD dwDirection)

{

mSrc = src;

mDirection = dwDirection;

mPos = 0;

}

override HRESULT QueryInterface(in IID\* riid, void\*\* pvObject)

{

if(queryInterface!(IEnumFORMATETC) (this, riid, pvObject))

return S\_OK;

return super.QueryInterface(riid, pvObject);

}

bool findValid()

{

while(mPos < mSrc.mCache.length)

{

if(mSrc.mCache[mPos].m\_nDataDir & mDirection)

return true;

mPos++;

}

return false;

}

override HRESULT Next(in ULONG celt,

/+[out, size\_is(celt), length\_is(\*pceltFetched )]+/ FORMATETC \*rgelt,

/+[out]+/ ULONG \*pceltFetched)

{

uint i;

for(i = 0; i < celt; i++)

if(findValid())

rgelt[i] = mSrc.mCache[mPos++].m\_formatEtc;

else

break;

if(pceltFetched)

\*pceltFetched = i;

return i < celt ? S\_FALSE : S\_OK;

}

override HRESULT Skip(in ULONG celt)

{

for(uint i = 0; i < celt; i++)

if(findValid())

mPos++;

else

return E\_FAIL;

return S\_OK;

}

override HRESULT Reset()

{

mPos = 0;

return S\_OK;

}

override HRESULT Clone(/+[out]+/ IEnumFORMATETC \*ppenum)

{

\*ppenum = addref(newCom!CEnumFormatEtc(mSrc, mDirection));

return S\_OK;

}

OleDataSource mSrc;

int mPos;

DWORD mDirection;

}

//---------------------------------------------------------------------------

HGLOBAL CopyGlobalMemory(HGLOBAL hDest, HGLOBAL hSource)

{

assert(hSource);

// make sure we have suitable hDest

uint nSize = GlobalSize(hSource);

assert(nSize < int.max);

if (!hDest)

{

hDest = GlobalAlloc(GMEM\_SHARE|GMEM\_MOVEABLE, nSize);

if (!hDest)

return null;

}

else if (nSize > GlobalSize(hDest))

{

// hDest is not large enough

return null;

}

// copy the bits

LPVOID lpSource = GlobalLock(hSource);

LPVOID lpDest = GlobalLock(hDest);

assert(lpDest && lpSource);

memcpy(lpDest, lpSource, nSize);

GlobalUnlock(hDest);

GlobalUnlock(hSource);

// success -- return hDest

return hDest;

}

//---------------------------------------------------------------------------

//---------------------------------------------------------------------------

BOOL \_CopyStgMedium(CLIPFORMAT cfFormat, STGMEDIUM\* lpDest, STGMEDIUM\* lpSource)

{

if (lpDest.tymed == TYMED\_NULL)

{

assert(lpSource.tymed != TYMED\_NULL);

switch (lpSource.tymed)

{

case TYMED\_ENHMF:

case TYMED\_HGLOBAL:

assert(HGLOBAL.sizeof == HENHMETAFILE.sizeof);

lpDest.tymed = lpSource.tymed;

lpDest.hGlobal = null;

break; // fall through to CopyGlobalMemory case

case TYMED\_ISTREAM:

lpDest.pstm = lpSource.pstm;

lpDest.pstm.AddRef();

lpDest.tymed = TYMED\_ISTREAM;

return TRUE;

case TYMED\_ISTORAGE:

lpDest.pstg = lpSource.pstg;

lpDest.pstg.AddRef();

lpDest.tymed = TYMED\_ISTORAGE;

return TRUE;

/+

case TYMED\_MFPICT:

{

// copy LPMETAFILEPICT struct + embedded HMETAFILE

HGLOBAL hDest = CopyGlobalMemory(null, lpSource.hGlobal);

if (hDest == null)

return FALSE;

LPMETAFILEPICT lpPict = cast(LPMETAFILEPICT)GlobalLock(hDest);

ASSERT(lpPict != null);

lpPict.hMF = CopyMetaFile(lpPict.hMF, null);

if (lpPict.hMF == null)

{

GlobalUnlock(hDest);

GlobalFree(hDest);

return FALSE;

}

GlobalUnlock(hDest);

// fill STGMEDIUM struct

lpDest.hGlobal = hDest;

lpDest.tymed = TYMED\_MFPICT;

}

return TRUE;

case TYMED\_GDI:

lpDest.tymed = TYMED\_GDI;

lpDest.hGlobal = null;

break;

case TYMED\_FILE:

{

USES\_CONVERSION;

lpDest.tymed = TYMED\_FILE;

ASSERT(lpSource.lpszFileName != null);

UINT cbSrc = ocslen(lpSource.lpszFileName);

LPOLESTR szFileName = cast(LPOLESTR)CoTaskMemAlloc((cbSrc+1)\*sizeof(OLECHAR));

lpDest.lpszFileName = szFileName;

if (szFileName == null)

return FALSE;

memcpy(szFileName, lpSource.lpszFileName, (cbSrc+1)\*sizeof(OLECHAR));

return TRUE;

}

+/

// unable to create + copy other TYMEDs

default:

return FALSE;

}

}

assert(lpDest.tymed == lpSource.tymed);

switch (lpSource.tymed)

{

case TYMED\_HGLOBAL:

{

HGLOBAL hDest = CopyGlobalMemory(lpDest.hGlobal, lpSource.hGlobal);

if (hDest == null)

return FALSE;

lpDest.hGlobal = hDest;

}

return TRUE;

/+

case TYMED\_ISTREAM:

{

ASSERT(lpDest.pstm != null);

ASSERT(lpSource.pstm != null);

// get the size of the source stream

STATSTG stat;

if (lpSource.pstm.Stat(&stat, STATFLAG\_NONAME) != S\_OK)

{

// unable to get size of source stream

return FALSE;

}

ASSERT(stat.pwcsName == null);

// always seek to zero before copy

LARGE\_INTEGER zero = { 0, 0 };

lpDest.pstm.Seek(zero, STREAM\_SEEK\_SET, null);

lpSource.pstm.Seek(zero, STREAM\_SEEK\_SET, null);

// copy source to destination

if (lpSource.pstm.CopyTo(lpDest.pstm, stat.cbSize,

null, null) != null)

{

// copy from source to dest failed

return FALSE;

}

// always seek to zero after copy

lpDest.pstm.Seek(zero, STREAM\_SEEK\_SET, null);

lpSource.pstm.Seek(zero, STREAM\_SEEK\_SET, null);

}

return TRUE;

case TYMED\_ISTORAGE:

{

ASSERT(lpDest.pstg != null);

ASSERT(lpSource.pstg != null);

// just copy source to destination

if (lpSource.pstg.CopyTo(0, null, null, lpDest.pstg) != S\_OK)

return FALSE;

}

return TRUE;

case TYMED\_FILE:

{

USES\_CONVERSION;

ASSERT(lpSource.lpszFileName != null);

ASSERT(lpDest.lpszFileName != null);

return CopyFile(OLE2T(lpSource.lpszFileName), OLE2T(lpDest.lpszFileName), FALSE);

}

case TYMED\_ENHMF:

case TYMED\_GDI:

{

ASSERT(sizeof(HGLOBAL) == sizeof(HENHMETAFILE));

// with TYMED\_GDI cannot copy into existing HANDLE

if (lpDest.hGlobal != null)

return FALSE;

// otherwise, use OleDuplicateData for the copy

lpDest.hGlobal = OleDuplicateData(lpSource.hGlobal, cfFormat, 0);

if (lpDest.hGlobal == null)

return FALSE;

}

return TRUE;

+/

// other TYMEDs cannot be copied

default:

return FALSE;

}

}

//---------------------------------------------------------------------------

// Helper for creating default FORMATETC from cfFormat

//---------------------------------------------------------------------------

FORMATETC\* \_FillFormatEtc(FORMATETC\* lpFormatEtc, CLIPFORMAT cfFormat, FORMATETC\* lpFormatEtcFill)

{

if (lpFormatEtc is null && cfFormat != 0)

{

lpFormatEtc = lpFormatEtcFill;

lpFormatEtc.cfFormat = cfFormat;

lpFormatEtc.ptd = null;

lpFormatEtc.dwAspect = DVASPECT\_CONTENT;

lpFormatEtc.lindex = -1;

lpFormatEtc.tymed = -1;

}

return lpFormatEtc;

}

// This file is part of Visual D

//

// Visual D integrates the D programming language into Visual Studio

// Copyright (c) 2010 by Rainer Schuetze, All Rights Reserved

//

// Distributed under the Boost Software License, Version 1.0.

// See accompanying file LICENSE\_1\_0.txt or copy at <http://www.boost.org/LICENSE_1_0.txt>

module visuald.pkgutil;

import visuald.hierutil;

import visuald.comutil;

import visuald.logutil;

import visuald.dpackage;

import std.conv;

import std.utf;

import sdk.vsi.vsshell;

void showStatusBarText(wstring txt)

{

auto pIVsStatusbar = queryService!(IVsStatusbar);

if(pIVsStatusbar)

{

scope(exit) release(pIVsStatusbar);

pIVsStatusbar.SetText((txt ~ "\0"w).ptr);

}

}

void showStatusBarText(string txt)

{

showStatusBarText(to!wstring(txt));

}

void deleteVisualDOutputPane()

{

auto win = queryService!(IVsOutputWindow)();

if(!win)

return;

scope(exit) release(win);

win.DeletePane(&g\_outputPaneCLSID);

}

void clearOutputPane()

{

auto win = queryService!(IVsOutputWindow)();

if(!win)

return;

scope(exit) release(win);

IVsOutputWindowPane pane;

if(win.GetPane(&g\_outputPaneCLSID, &pane) == S\_OK && pane)

pane.Clear();

release(pane);

}

IVsOutputWindowPane getVisualDOutputPane()

{

auto win = queryService!(IVsOutputWindow)();

if(!win)

return null;

scope(exit) release(win);

IVsOutputWindowPane pane;

if(win.GetPane(&g\_outputPaneCLSID, &pane) == S\_OK && pane)

return pane;

if(win.CreatePane(&g\_outputPaneCLSID, "Visual D", false, true) == S\_OK)

if(win.GetPane(&g\_outputPaneCLSID, &pane) == S\_OK && pane)

return pane;

if(win.GetPane(&GUID\_BuildOutputWindowPane, &pane) != S\_OK || !pane)

return null;

return pane;

}

IVsOutputWindowPane getBuildOutputPane()

{

auto win = queryService!(IVsOutputWindow)();

if(!win)

return null;

scope(exit) release(win);

IVsOutputWindowPane pane;

if(win.GetPane(&GUID\_BuildOutputWindowPane, &pane) != S\_OK || !pane)

return null;

return pane;

}

class OutputPaneBuffer

{

static shared(string) buffer;

static void push(string msg)

{

synchronized(OutputPaneBuffer.classinfo)

buffer ~= msg;

}

static string pop()

{

string msg;

synchronized(OutputPaneBuffer.classinfo)

{

msg = buffer;

buffer = buffer.init;

}

return msg;

}

static void flush()

{

if(buffer.length)

{

string msg = pop();

writeToBuildOutputPane(msg);

}

}

}

void writeToBuildOutputPane(string msg)

{

if(IVsOutputWindowPane pane = getVisualDOutputPane())

{

scope(exit) release(pane);

pane.Activate();

pane.OutputString(\_toUTF16z(msg));

}

else

OutputPaneBuffer.push(msg);

}

bool OutputErrorString(string msg)

{

if (IVsOutputWindowPane pane = getVisualDOutputPane())

{

scope(exit) release(pane);

pane.OutputString(toUTF16z(msg));

}

return false;

}

bool tryWithExceptionToBuildOutputPane(T)(T dg, string errInfo = "")

{

try

{

dg();

return true;

}

catch(Exception e)

{

string msg = e.toString();

if(errInfo.length)

msg = errInfo ~ ": " ~ msg;

writeToBuildOutputPane(msg);

logCall("EXCEPTION: " ~ msg);

}

return false;

}

string browseFile(HWND parentHwnd, string title, string filter, string initdir = null)

{

if (auto pIVsUIShell = ComPtr!(IVsUIShell)(queryService!(IVsUIShell), false))

{

wchar[260] fileName;

fileName[0] = 0;

VSOPENFILENAMEW ofn;

ofn.lStructSize = ofn.sizeof;

ofn.hwndOwner = parentHwnd;

ofn.pwzDlgTitle = toUTF16z(title);

ofn.pwzFileName = fileName.ptr;

ofn.nMaxFileName = fileName.length;

ofn.pwzInitialDir = toUTF16z(initdir);

ofn.pwzFilter = toUTF16z(filter);

if (pIVsUIShell.GetOpenFileNameViaDlg(&ofn) == S\_OK)

return to!string(fileName);

}

return null;

}

string browseDirectory(HWND parentHwnd, string title, string initdir = null)

{

if (auto pIVsUIShell = ComPtr!(IVsUIShell)(queryService!(IVsUIShell), false))

{

wchar[260] dirName;

dirName[0] = 0;

VSBROWSEINFOW bi;

bi.lStructSize = bi.sizeof;

bi.hwndOwner = parentHwnd;

bi.pwzDlgTitle = toUTF16z(title);

bi.pwzDirName = dirName.ptr;

bi.nMaxDirName = dirName.length;

bi.pwzInitialDir = toUTF16z(initdir);

if (pIVsUIShell.GetDirectoryViaBrowseDlg(&bi) == S\_OK)

return to!string(dirName);

}

return null;

}

///////////////////////////////////////////////////////////////////////

// version = DEBUG\_GC;

version(DEBUG\_GC)

{

import rsgc.gc;

import rsgc.gcx;

import rsgc.gcstats;

import std.string;

void writeGCStatsToOutputPane()

{

GCStats stats =        gc\_stats();

writeToBuildOutputPane(format("numpools = %s, poolsize = %s, usedsize = %s, freelistsize = %s\n",

stats.numpools, stats.poolsize, stats.usedsize, stats.freelistsize));

writeToBuildOutputPane(format("pool[B\_16] = %s\n", stats.numpool[B\_16] ));

writeToBuildOutputPane(format("pool[B\_32] = %s\n", stats.numpool[B\_32] ));

writeToBuildOutputPane(format("pool[B\_64] = %s\n", stats.numpool[B\_64] ));

writeToBuildOutputPane(format("pool[B\_128] = %s\n", stats.numpool[B\_128] ));

writeToBuildOutputPane(format("pool[B\_256] = %s\n", stats.numpool[B\_256] ));

writeToBuildOutputPane(format("pool[B\_512] = %s\n", stats.numpool[B\_512] ));

writeToBuildOutputPane(format("pool[B\_1024] = %s\n", stats.numpool[B\_1024] ));

writeToBuildOutputPane(format("pool[B\_2048] = %s\n", stats.numpool[B\_2048] ));

writeToBuildOutputPane(format("pool[B\_PAGE] = %s\n", stats.numpool[B\_PAGE] ));

writeToBuildOutputPane(format("pool[B\_PAGE+] = %s\n", stats.numpool[B\_PAGEPLUS]));

writeToBuildOutputPane(format("pool[B\_FREE] = %s\n", stats.numpool[B\_FREE] ));

writeToBuildOutputPane(format("pool[B\_UNCOM] = %s\n", stats.numpool[B\_UNCOMMITTED]));

writeClasses();

}

extern extern(C) \_\_gshared ModuleInfo\*[] \_moduleinfo\_array;

void writeClasses()

{

foreach(mi; \_moduleinfo\_array)

{

auto classes = mi.localClasses();

foreach(c; classes)

{

string flags;

if(c.m\_flags & 1) flags ~= " IUnknown";

if(c.m\_flags & 2) flags ~= " NoGC";

if(c.m\_flags & 4) flags ~= " OffTI";

if(c.m\_flags & 8) flags ~= " Constr";

if(c.m\_flags & 16) flags ~= " xgetM";

if(c.m\_flags & 32) flags ~= " tinfo";

if(c.m\_flags & 64) flags ~= " abstract";

writeToBuildOutputPane(text(c.name, ": ", c.init.length, " bytes, flags: ", flags, "\n"));

foreach(m; c.getMembers([]))

{

auto cm = cast() m;

writeToBuildOutputPane(text(" ", cm.name(), "\n"));

}

}

}

}

}

///////////////////////////////////////////////////////////////////////

HRESULT GetSelectionForward(IVsTextView view, int\*startLine, int\*startCol, int\*endLine, int\*endCol)

{

HRESULT hr = view.GetSelection(startLine, startCol, endLine, endCol);

if(FAILED(hr))

return hr;

if(\*startLine > \*endLine)

{

std.algorithm.swap(\*startLine, \*endLine);

std.algorithm.swap(\*startCol, \*endCol);

}

else if(\*startLine == \*endLine && \*startCol > \*endCol)

std.algorithm.swap(\*startCol, \*endCol);

return hr;

}

///////////////////////////////////////////////////////////////////////

// Hardware Breakpoint Functions

enum

{

HWBRK\_TYPE\_CODE,

HWBRK\_TYPE\_READWRITE,

HWBRK\_TYPE\_WRITE,

}

alias int HWBRK\_TYPE;

enum

{

HWBRK\_SIZE\_1,

HWBRK\_SIZE\_2,

HWBRK\_SIZE\_4,

HWBRK\_SIZE\_8,

}

alias int HWBRK\_SIZE;

struct HWBRK

{

public:

void\* a;

HANDLE hT;

HWBRK\_TYPE Type;

HWBRK\_SIZE Size;

HANDLE hEv;

int iReg;

int Opr;

bool SUCC;

}

void SetBits(ref uint dw, int lowBit, int bits, int newValue)

{

DWORD\_PTR mask = (1 << bits) - 1;

dw = (dw & ~(mask << lowBit)) | (newValue << lowBit);

}

extern(Windows) DWORD thSuspend(LPVOID lpParameter)

{

HWBRK\* h = cast(HWBRK\*)lpParameter;

int j = 0;

int y = 0;

j = SuspendThread(h.hT);

y = GetLastError();

h.SUCC = th(h);

j = ResumeThread(h.hT);

y = GetLastError();

SetEvent(h.hEv);

return 0;

}

bool th(HWBRK\* h)

{

int j = 0;

int y = 0;

CONTEXT ct;

ct.ContextFlags = CONTEXT\_DEBUG\_REGISTERS;

j = GetThreadContext(h.hT,&ct);

y = GetLastError();

int FlagBit = 0;

bool Dr0Busy = false;

bool Dr1Busy = false;

bool Dr2Busy = false;

bool Dr3Busy = false;

if (ct.Dr7 & 1)

Dr0Busy = true;

if (ct.Dr7 & 4)

Dr1Busy = true;

if (ct.Dr7 & 16)

Dr2Busy = true;

if (ct.Dr7 & 64)

Dr3Busy = true;

if (h.Opr == 1)

{

// Remove

if (h.iReg == 0)

{

FlagBit = 0;

ct.Dr0 = 0;

Dr0Busy = false;

}

if (h.iReg == 1)

{

FlagBit = 2;

ct.Dr1 = 0;

Dr1Busy = false;

}

if (h.iReg == 2)

{

FlagBit = 4;

ct.Dr2 = 0;

Dr2Busy = false;

}

if (h.iReg == 3)

{

FlagBit = 6;

ct.Dr3 = 0;

Dr3Busy = false;

}

ct.Dr7 &= ~(1 << FlagBit);

}

else

{

if (!Dr0Busy)

{

h.iReg = 0;

ct.Dr0 = cast(DWORD)h.a;

Dr0Busy = true;

}

else if (!Dr1Busy)

{

h.iReg = 1;

ct.Dr1 = cast(DWORD)h.a;

Dr1Busy = true;

}

else if (!Dr2Busy)

{

h.iReg = 2;

ct.Dr2 = cast(DWORD)h.a;

Dr2Busy = true;

}

else if (!Dr3Busy)

{

h.iReg = 3;

ct.Dr3 = cast(DWORD)h.a;

Dr3Busy = true;

}

else

{

return false;

}

ct.Dr6 = 0;

int st = 0;

if (h.Type == HWBRK\_TYPE\_CODE)

st = 0;

if (h.Type == HWBRK\_TYPE\_READWRITE)

st = 3;

if (h.Type == HWBRK\_TYPE\_WRITE)

st = 1;

int le = 0;

if (h.Size == HWBRK\_SIZE\_1)

le = 0;

if (h.Size == HWBRK\_SIZE\_2)

le = 1;

if (h.Size == HWBRK\_SIZE\_4)

le = 3;

if (h.Size == HWBRK\_SIZE\_8)

le = 2;

SetBits(ct.Dr7, 16 + h.iReg\*4, 2, st);

SetBits(ct.Dr7, 18 + h.iReg\*4, 2, le);

SetBits(ct.Dr7, h.iReg\*2,1,1);

}

ct.ContextFlags = CONTEXT\_DEBUG\_REGISTERS;

j = SetThreadContext(h.hT,&ct);

y = GetLastError();

ct.ContextFlags = CONTEXT\_DEBUG\_REGISTERS;

j = GetThreadContext(h.hT,&ct);

y = GetLastError();

return true;

}

extern(C)

HANDLE SetHardwareBreakpoint(HANDLE hThread,HWBRK\_TYPE Type,HWBRK\_SIZE Size,void\* s)

{

//HWBRK\* h = new HWBRK;

HWBRK h;

h.a = s;

h.Size = Size;

h.Type = Type;

h.hT = hThread;

if (hThread == GetCurrentThread())

{

DWORD pid = GetCurrentThreadId();

h.hT = OpenThread(THREAD\_SUSPEND\_RESUME|THREAD\_GET\_CONTEXT|THREAD\_SET\_CONTEXT,0,pid);

}

version(none)

{

h.hEv = CreateEvent(null,0,0,null);

h.Opr = 0; // Set Break

HANDLE hY = CreateThread(null,0,&thSuspend,cast(LPVOID)&h,0,null);

WaitForSingleObject(h.hEv,INFINITE);

CloseHandle(h.hEv);

h.hEv = null;

}

else

{

th(&h);

}

if (hThread == GetCurrentThread())

{

CloseHandle(h.hT);

}

h.hT = hThread;

//        if (!h.SUCC)

{

//                delete h;

return null;

}

//        return cast(HANDLE)h;

}

extern(C)

bool RemoveHardwareBreakpoint(HANDLE hBrk)

{

HWBRK\* h = cast(HWBRK\*)hBrk;

if (!h)

return false;

bool C = false;

if (h.hT == GetCurrentThread())

{

DWORD pid = GetCurrentThreadId();

h.hT = OpenThread(THREAD\_ALL\_ACCESS,0,pid);

C = true;

}

h.hEv = CreateEvent(null,0,0,null);

h.Opr = 1; // Remove Break

HANDLE hY = CreateThread(null,0,&thSuspend,cast(LPVOID)h,0,null);

WaitForSingleObject(h.hEv,INFINITE);

CloseHandle(h.hEv);

h.hEv = null;

if (C)

{

CloseHandle(h.hT);

}

delete h;

return true;

}

//import pkgutil;

import sdk.port.base;

import visuald.dllmain;

void setHWBreakpopints()

{

char[] data = new char[16];

HANDLE hnd;

void\* addr1 = data.ptr - 0x71bffc0 + 0x71bf720;

void\* addr2 = data.ptr - 0x71bffc0 + 0x71bf8a0;

void\* addr3 = data.ptr - 0x71eff60 + 0x71e6420;

void\* addr4 = data.ptr - 0x71eff60 + 0x71e6440;

hnd = SetHardwareBreakpoint(GetCurrentThread(), HWBRK\_TYPE\_WRITE, HWBRK\_SIZE\_4, addr1);

//hnd = SetHardwareBreakpoint(GetCurrentThread(), HWBRK\_TYPE\_READWRITE, HWBRK\_SIZE\_4, addr2);

//hnd = SetHardwareBreakpoint(GetCurrentThread(), HWBRK\_TYPE\_WRITE, HWBRK\_SIZE\_4, addr3);

//hnd = SetHardwareBreakpoint(GetCurrentThread(), HWBRK\_TYPE\_WRITE, HWBRK\_SIZE\_4, addr4);

addr1 = null;

addr2 = null;

addr3 = null;

addr4 = null;

}

// This file is part of Visual D

//

// Visual D integrates the D programming language into Visual Studio

// Copyright (c) 2010 by Rainer Schuetze, All Rights Reserved

//

// Distributed under the Boost Software License, Version 1.0.

// See accompanying file LICENSE\_1\_0.txt or copy at <http://www.boost.org/LICENSE_1_0.txt>

module visuald.profiler;

import visuald.windows;

import visuald.winctrl;

import visuald.comutil;

import visuald.dimagelist;

import visuald.register;

import visuald.hierutil;

import visuald.logutil;

import visuald.stringutil;

import visuald.pkgutil;

import visuald.dpackage;

import visuald.intellisense;

import visuald.config;

import visuald.wmmsg;

import sdk.win32.commctrl;

import sdk.vsi.vsshell;

import sdk.vsi.vsshell80;

import stdext.string;

import std.conv;

import std.utf;

import std.stdio;

import std.string;

import std.algorithm;

import std.file;

import std.path;

import core.demangle;

private IVsWindowFrame sWindowFrame;

private        ProfilePane sProfilePane;

bool showProfilerWindow()

{

if(!sWindowFrame)

{

auto pIVsUIShell = ComPtr!(IVsUIShell)(queryService!(IVsUIShell), false);

if(!pIVsUIShell)

return false;

sProfilePane = newCom!ProfilePane;

const(wchar)\* caption = "Visual D Profiler"w.ptr;

HRESULT hr;

hr = pIVsUIShell.CreateToolWindow(CTW\_fInitNew, 0, sProfilePane,

&GUID\_NULL, &g\_profileWinCLSID, &GUID\_NULL,

null, caption, null, &sWindowFrame);

if(!SUCCEEDED(hr))

{

sProfilePane = null;

return false;

}

}

if(FAILED(sWindowFrame.Show()))

return false;

BOOL fHandled;

sProfilePane.\_OnSetFocus(0, 0, 0, fHandled);

return fHandled != 0;

}

const int kColumnInfoVersion = 1;

const bool kToolBarAtTop = true;

const int kToolBarHeight = 24;

const int kPaneMargin = 0;

const int kBackMargin = 2;

const HDMIL\_PRIVATE = 0xf00d;

struct static\_COLUMNINFO

{

string displayName;

int fmt;

int cx;

}

struct COLUMNINFO

{

COLUMNID colid;

BOOL fVisible;

int cx;

};

enum COLUMNID

{

NONE = -1,

NAME,

CALLS,

TREETIME,

FUNCTIME,

CALLTIME,

MAX

}

const static\_COLUMNINFO[] s\_rgColumns =

[

//{ "none", LVCFMT\_LEFT, 80 },

{ "Function", LVCFMT\_LEFT, 80 },

{ "Calls", LVCFMT\_LEFT, 80 },

{ "Tree Time", LVCFMT\_LEFT, 80 },

{ "Func Time", LVCFMT\_LEFT, 80 },

{ "Call Time", LVCFMT\_LEFT, 80 },

];

const COLUMNINFO[] default\_Columns =

[

{ COLUMNID.NAME, true, 300 },

{ COLUMNID.CALLS, true, 100 },

{ COLUMNID.TREETIME, true, 100 },

{ COLUMNID.FUNCTIME, true, 100 },

{ COLUMNID.CALLTIME, true, 100 },

];

const static\_COLUMNINFO[] s\_rgFanInColumns =

[

//{ "none", LVCFMT\_LEFT, 80 },

{ "Caller", LVCFMT\_LEFT, 80 },

{ "Calls", LVCFMT\_LEFT, 80 },

];

const static\_COLUMNINFO[] s\_rgFanOutColumns =

[

//{ "none", LVCFMT\_LEFT, 80 },

{ "Callee", LVCFMT\_LEFT, 80 },

{ "Calls", LVCFMT\_LEFT, 80 },

];

const COLUMNINFO[] default\_FanColumns =

[

{ COLUMNID.NAME, true, 300 },

{ COLUMNID.CALLS, true, 100 },

];

struct INDEXQUERYPARAMS

{

COLUMNID colidSort;

bool fSortAscending;

COLUMNID colidGroup;

}

class ProfileWindowBack : Window

{

this(Window parent, ProfilePane pane)

{

mPane = pane;

super(parent);

}

override int WindowProc(HWND hWnd, uint uMsg, WPARAM wParam, LPARAM lParam)

{

BOOL fHandled;

LRESULT rc = mPane.\_WindowProc(hWnd, uMsg, wParam, lParam, fHandled);

if(fHandled)

return rc;

return super.WindowProc(hWnd, uMsg, wParam, lParam);

}

ProfilePane mPane;

}

class ProfilePane : DisposingComObject, IVsWindowPane

{

IServiceProvider mSite;

override HRESULT QueryInterface(in IID\* riid, void\*\* pvObject)

{

if(queryInterface!(IVsWindowPane) (this, riid, pvObject))

return S\_OK;

// avoid debug output

if(\*riid == IVsCodeWindow.iid || \*riid == IServiceProvider.iid || \*riid == IVsTextView.iid)

return E\_NOINTERFACE;

return super.QueryInterface(riid, pvObject);

}

override void Dispose()

{

mSite = release(mSite);

}

HRESULT SetSite(/+[in]+/ IServiceProvider pSP)

{

mixin(LogCallMix2);

mSite = release(mSite);

mSite = addref(pSP);

return S\_OK;

}

HRESULT CreatePaneWindow(in HWND hwndParent, in int x, in int y, in int cx, in int cy,

/+[out]+/ HWND \*hwnd)

{

mixin(LogCallMix2);

\_wndParent = new Window(hwndParent);

\_wndBack = new ProfileWindowBack(\_wndParent, this);

BOOL fHandled;

\_OnInitDialog(WM\_INITDIALOG, 0, 0, fHandled);

\_CheckSize();

\_wndBack.setVisible(true);

return S\_OK;

}

HRESULT GetDefaultSize(/+[out]+/ SIZE \*psize)

{

mixin(LogCallMix2);

psize.cx = 300;

psize.cy = 200;

return S\_OK;

}

HRESULT ClosePane()

{

mixin(LogCallMix2);

if(\_wndParent)

{

\_wndParent.Dispose();

\_wndParent = null;

\_wndBack = null;

\_wndFileWheel = null;

\_wndFuncList = null;

\_wndFuncListHdr = null;

\_wndFanInList = null;

\_wndFanOutList = null;

\_wndToolbar = null;

if(\_himlToolbar)

ImageList\_Destroy(\_himlToolbar);

\_lastResultsArray = null;

mDlgFont = deleteDialogFont(mDlgFont);

}

return S\_OK;

}

HRESULT LoadViewState(/+[in]+/ IStream pstream)

{

mixin(LogCallMix2);

return returnError(E\_NOTIMPL);

}

HRESULT SaveViewState(/+[in]+/ IStream pstream)

{

mixin(LogCallMix2);

return returnError(E\_NOTIMPL);

}

HRESULT TranslateAccelerator(MSG\* msg)

{

if(msg.message == WM\_TIMER)

\_CheckSize();

if(msg.message == WM\_TIMER || msg.message == WM\_SYSTIMER)

return E\_NOTIMPL; // do not flood debug output

logMessage("TranslateAccelerator", msg.hwnd, msg.message, msg.wParam, msg.lParam);

BOOL fHandled;

HRESULT hrRet = \_HandleMessage(msg.hwnd, msg.message, msg.wParam, msg.lParam, fHandled);

if(fHandled)

return hrRet;

return E\_NOTIMPL;

}

///////////////////////////////////////////////////////////////////

private:

Window \_wndParent;

ProfileWindowBack \_wndBack;

HFONT mDlgFont;

Text \_wndFileWheel;

ListView \_wndFuncList;

ListView \_wndFanInList;

ListView \_wndFanOutList;

Window \_wndFuncListHdr;

ToolBar \_wndToolbar;

HIMAGELIST \_himlToolbar;

ItemArray \_lastResultsArray; // remember to keep reference to ProfileItems referenced in list items

ProfileItemIndex \_spsii;

int \_lastSelectedItem;

BOOL \_fShowFanInOut;

BOOL \_fFullDecoration;

BOOL \_fAlternateRowColor;

BOOL \_closeOnReturn;

COLUMNINFO[] \_rgColumns;

INDEXQUERYPARAMS \_iqp;

COLORREF \_crAlternate;

static HINSTANCE getInstance() { return Widget.getInstance(); }

int \_WindowProc(HWND hWnd, uint uMsg, WPARAM wParam, LPARAM lParam, ref BOOL fHandled)

{

if(uMsg != WM\_NOTIFY)

logMessage("\_WindowProc", hWnd, uMsg, wParam, lParam);

return \_HandleMessage(hWnd, uMsg, wParam, lParam, fHandled);

}

int \_HandleMessage(HWND hWnd, uint uMsg, WPARAM wParam, LPARAM lParam, ref BOOL fHandled)

{

switch(uMsg)

{

case WM\_CREATE:

case WM\_INITDIALOG:

return \_OnInitDialog(uMsg, wParam, lParam, fHandled);

case WM\_NCCALCSIZE:

return \_OnCalcSize(uMsg, wParam, lParam, fHandled);

case WM\_SIZE:

return \_OnSize(uMsg, wParam, lParam, fHandled);

case WM\_NCACTIVATE:

case WM\_SETFOCUS:

return \_OnSetFocus(uMsg, wParam, lParam, fHandled);

case WM\_CONTEXTMENU:

return \_OnContextMenu(uMsg, wParam, lParam, fHandled);

case WM\_DESTROY:

return \_OnDestroy(uMsg, wParam, lParam, fHandled);

case WM\_KEYDOWN:

case WM\_SYSKEYDOWN:

return \_OnKeyDown(uMsg, wParam, lParam, fHandled);

case WM\_COMMAND:

ushort id = LOWORD(wParam);

ushort code = HIWORD(wParam);

if(id == IDC\_FILEWHEEL && code == EN\_CHANGE)

return \_OnFileWheelChanged(id, code, hWnd, fHandled);

if(code == BN\_CLICKED)

{

switch(id)

{

case IDOK:

return \_OnOpenSelectedItem(code, id, hWnd, fHandled);

case IDR\_ALTERNATEROWCOLOR:

case IDR\_GROUPBYKIND:

case IDR\_CLOSEONRETURN:

case IDR\_FANINOUT:

case IDR\_FULLDECO:

case IDR\_REMOVETRACE:

case IDR\_SETTRACE:

case IDR\_REFRESH:

return \_OnCheckBtnClicked(code, id, hWnd, fHandled);

default:

break;

}

}

break;

case WM\_NOTIFY:

NMHDR\* nmhdr = cast(NMHDR\*)lParam;

if(nmhdr.idFrom == IDC\_FILELIST)

{

switch(nmhdr.code)

{

case LVN\_GETDISPINFO:

return \_OnFileListGetDispInfo(wParam, nmhdr, fHandled);

case LVN\_COLUMNCLICK:

return \_OnFileListColumnClick(wParam, nmhdr, fHandled);

case LVN\_DELETEITEM:

return \_OnFileListDeleteItem(wParam, nmhdr, fHandled);

case LVN\_ITEMCHANGED:

return \_OnFileListItemChanged(wParam, nmhdr, fHandled);

case NM\_DBLCLK:

return \_OnFileListDblClick(wParam, nmhdr, fHandled);

case NM\_CUSTOMDRAW:

return \_OnFileListCustomDraw(wParam, nmhdr, fHandled);

default:

break;

}

}

if(nmhdr.idFrom == IDC\_FANINLIST && nmhdr.code == NM\_DBLCLK)

return \_OnFanInOutListDblClick(true, nmhdr, fHandled);

if(nmhdr.idFrom == IDC\_FANOUTLIST && nmhdr.code == NM\_DBLCLK)

return \_OnFanInOutListDblClick(false, nmhdr, fHandled);

if (nmhdr.idFrom == IDC\_FILELISTHDR && nmhdr.code == HDN\_ITEMCHANGED)

return \_OnFileListHdrItemChanged(wParam, nmhdr, fHandled);

if (nmhdr.idFrom == IDC\_TOOLBAR && nmhdr.code == TBN\_GETINFOTIP)

return \_OnToolbarGetInfoTip(wParam, nmhdr, fHandled);

break;

default:

break;

}

return 0;

}

public this()

{

\_fAlternateRowColor = true;

\_closeOnReturn = true;

\_spsii = new ProfileItemIndex();

\_rgColumns = default\_Columns.dup;

\_iqp.colidSort = COLUMNID.NAME;

\_iqp.fSortAscending = true;

\_iqp.colidGroup = COLUMNID.NONE;

}

void \_MoveSelection(BOOL fDown)

{

// Get the current selection

int iSel = \_wndFuncList.SendMessage(LVM\_GETNEXTITEM, cast(WPARAM)-1, LVNI\_SELECTED);

int iCnt = \_wndFuncList.SendMessage(LVM\_GETITEMCOUNT);

if(iSel == 0 && !fDown)

return;

if(iSel == iCnt - 1 && fDown)

return;

\_UpdateSelection(iSel, fDown ? iSel+1 : iSel-1);

}

void \_UpdateSelection(int from, int to)

{

LVITEM lvi;

lvi.iItem = from;

lvi.mask = LVIF\_STATE;

lvi.stateMask = LVIS\_SELECTED | LVIS\_FOCUSED;

lvi.state = 0;

\_wndFuncList.SendItemMessage(LVM\_SETITEM, lvi);

lvi.iItem = to;

lvi.mask = LVIF\_STATE;

lvi.stateMask = LVIS\_SELECTED | LVIS\_FOCUSED;

lvi.state = LVIS\_SELECTED | LVIS\_FOCUSED;

\_wndFuncList.SendItemMessage(LVM\_SETITEM, lvi);

\_wndFuncList.SendMessage(LVM\_ENSUREVISIBLE, lvi.iItem, FALSE);

}

HRESULT \_PrepareFileListForResults(in ItemArray puaResults)

{

\_wndFuncList.SendMessage(LVM\_DELETEALLITEMS);

\_wndFuncList.SendMessage(LVM\_REMOVEALLGROUPS);

HIMAGELIST himl = LoadImageList(getInstance(), MAKEINTRESOURCEA(BMP\_DIMAGELIST), 16, 16);

if(himl)

\_wndFuncList.SendMessage(LVM\_SETIMAGELIST, LVSIL\_SMALL, cast(LPARAM)himl);

HRESULT hr = S\_OK;

BOOL fEnableGroups = \_iqp.colidGroup != COLUMNID.NONE;

if (fEnableGroups)

{

DWORD cGroups = puaResults.GetCount();

// Don't enable groups if there is only 1

if (cGroups <= 1)

{

fEnableGroups = FALSE;

}

}

if (SUCCEEDED(hr))

{

hr = \_wndFuncList.SendMessage(LVM\_ENABLEGROUPVIEW, fEnableGroups) == -1 ? E\_FAIL : S\_OK;

}

return hr;

}

HRESULT \_AddItemsToFileList(int iGroupId, in ItemArray pua)

{

LVITEM lvi;

lvi.pszText = LPSTR\_TEXTCALLBACK;

lvi.iItem = cast(int)\_wndFuncList.SendMessage(LVM\_GETITEMCOUNT);

DWORD cItems = pua.GetCount();

HRESULT hr = S\_OK;

for (DWORD i = 0; i < cItems && SUCCEEDED(hr); i++)

{

if(ProfileItem spsi = pua.GetItem(i))

{

for (int iCol = COLUMNID.NAME; iCol < COLUMNID.MAX; iCol++)

{

lvi.iSubItem = iCol;

if (iCol != COLUMNID.NAME)

{

lvi.mask = LVIF\_TEXT;

}

else

{

lvi.mask = LVIF\_PARAM | LVIF\_TEXT | LVIF\_IMAGE;

lvi.iGroupId = iGroupId;

lvi.lParam = cast(LPARAM)cast(void\*)spsi;

lvi.iImage = spsi.GetIconIndex();

if (iGroupId != -1)

{

lvi.mask |= LVIF\_GROUPID;

lvi.iGroupId = iGroupId;

}

}

if (\_wndFuncList.SendItemMessage(LVM\_INSERTITEM, lvi) != -1 && iCol == COLUMNID.NAME)

{

//spsi.detach();

}

}

spsi = null;

}

lvi.iItem++;

}

return hr;

}

HRESULT \_AddGroupToFileList(int iGroupId, in ProfileItemGroup psig)

{

LVGROUP lvg;

lvg.cbSize = lvg.sizeof;

lvg.mask = LVGF\_ALIGN | LVGF\_HEADER | LVGF\_GROUPID | LVGF\_STATE;

lvg.uAlign = LVGA\_HEADER\_LEFT;

lvg.iGroupId = iGroupId;

lvg.pszHeader = \_toUTF16z(psig.GetName());

lvg.state = LVGS\_NORMAL;

HRESULT hr = \_wndFuncList.SendMessage(LVM\_INSERTGROUP, cast(WPARAM)-1, cast(LPARAM)&lvg) != -1 ? S\_OK : E\_FAIL;

if (SUCCEEDED(hr))

{

const(ItemArray) spItems = psig.GetItems();

if(spItems)

{

hr = \_AddItemsToFileList(iGroupId, spItems);

}

}

return hr;

}

HRESULT \_RefreshFileList()

{

mixin(LogCallMix);

\_wndFuncList.SetRedraw(FALSE);

HRESULT hr = S\_OK;

string strWordWheel = \_wndFileWheel.GetWindowText();

ItemArray spResultsArray;

hr = \_spsii.Update(strWordWheel, &\_iqp, &spResultsArray);

if (SUCCEEDED(hr))

{

hr = \_PrepareFileListForResults(spResultsArray);

if (SUCCEEDED(hr))

{

if (\_iqp.colidGroup != COLUMNID.NONE)

{

DWORD cGroups = spResultsArray.GetCount();

for (DWORD iGroup = 0; iGroup < cGroups && SUCCEEDED(hr); iGroup++)

{

if(ProfileItemGroup spsig = spResultsArray.GetGroup(iGroup))

{

hr = \_AddGroupToFileList(iGroup, spsig);

}

}

}

else

{

hr = \_AddItemsToFileList(-1, spResultsArray);

}

}

\_lastResultsArray = spResultsArray;

}

if (SUCCEEDED(hr))

{

// Select the first item

LVITEM lviSelect;

lviSelect.mask = LVIF\_STATE;

lviSelect.iItem = 0;

lviSelect.state = LVIS\_SELECTED | LVIS\_FOCUSED;

lviSelect.stateMask = LVIS\_SELECTED | LVIS\_FOCUSED;

\_wndFuncList.SendItemMessage(LVM\_SETITEM, lviSelect);

}

\_wndFuncList.SetRedraw(TRUE);

\_wndFuncList.InvalidateRect(null, FALSE);

return hr;

}

string \_demangle(string txt, bool fullDeco)

{

static if(\_\_traits(compiles, (){uint p; decodeDmdString("", p);}))

uint p = 0;

else

int p = 0; // until dmd 2.056

version(all) // debug // allow std 2.052 in debug builds

enum hasTypeArg = \_\_traits(compiles, { demangle("",true); });

else // ensure patched runtime in release

enum hasTypeArg = true;

static bool isDSymbolChar(char c)

{

if (('a' <= c && c <= 'z') || ('A' <= c && c <= 'Z') || ('0' <= c && c <= '9') || c == '\_')

return true;

return (0x80 & c) != 0; // any compressed or unicode symbol

}

for (size\_t i = 0; i < txt.length; i++)

if (!isDSymbolChar(txt[i]))

return txt;

txt = decodeDmdString(txt, p);

if(txt.length > 2 && txt[0] == '\_' && txt[1] == 'D')

{

static if(hasTypeArg)

txt = to!string(demangle(txt, fullDeco));

else

{

pragma(msg, text(\_\_FILE\_\_, "(", \_\_LINE\_\_, "): profiler.\_demangle uses compatibility mode, this won't allow disabling type info"));

txt = to!string(demangle(txt));

}

}

return txt;

}

void \_InsertFanInOut(ListView lv, Fan fan)

{

LVITEM lvi;

lvi.pszText = \_toUTF16z(\_demangle(fan.func, \_fFullDecoration != 0));

lvi.iItem = cast(int)lv.SendMessage(LVM\_GETITEMCOUNT);

lvi.mask = LVIF\_TEXT;

lv.SendItemMessage(LVM\_INSERTITEM, lvi);

lvi.pszText = \_toUTF16z(to!string(fan.calls));

lvi.iSubItem = 1;

lvi.mask = LVIF\_TEXT;

lv.SendItemMessage(LVM\_SETITEM, lvi);

}

void RefreshFanInOutList(ProfileItem psi)

{

if(!psi || !\_fShowFanInOut)

return;

\_wndFanInList.SendMessage(LVM\_DELETEALLITEMS);

\_wndFanOutList.SendMessage(LVM\_DELETEALLITEMS);

foreach(fan; psi.mFanIn)

\_InsertFanInOut(\_wndFanInList, fan);

foreach(fan; psi.mFanOut)

\_InsertFanInOut(\_wndFanOutList, fan);

}

// Special icon dimensions for the sort direction indicator

enum int c\_cxSortIcon = 7;

enum int c\_cySortIcon = 6;

HRESULT \_CreateSortImageList(out HIMAGELIST phiml)

{

// Create an image list for the sort direction indicators

HIMAGELIST himl = ImageList\_Create(c\_cxSortIcon, c\_cySortIcon, ILC\_COLORDDB | ILC\_MASK, 2, 1);

HRESULT hr = himl ? S\_OK : E\_OUTOFMEMORY;

if (SUCCEEDED(hr))

{

HICON hicn = cast(HICON)LoadImage(getInstance(), MAKEINTRESOURCE(IDI\_DESCENDING), IMAGE\_ICON, c\_cxSortIcon, c\_cySortIcon, LR\_DEFAULTCOLOR | LR\_SHARED);

hr = hicn ? S\_OK : HResultFromLastError();

if (SUCCEEDED(hr))

{

hr = ImageList\_ReplaceIcon(himl, -1, hicn) != -1 ? S\_OK : E\_FAIL;

if (SUCCEEDED(hr))

{

hicn = cast(HICON)LoadImage(getInstance(), MAKEINTRESOURCE(IDI\_ASCENDING), IMAGE\_ICON, c\_cxSortIcon, c\_cySortIcon, LR\_DEFAULTCOLOR | LR\_SHARED);

hr = hicn ? S\_OK : HResultFromLastError();

if (SUCCEEDED(hr))

{

hr = ImageList\_ReplaceIcon(himl, -1, hicn) != -1 ? S\_OK : E\_FAIL;

if (SUCCEEDED(hr))

{

phiml = himl;

himl = null;

}

}

}

}

if (himl)

{

ImageList\_Destroy(himl);

}

}

return hr;

}

HRESULT \_AddSortIcon(int iIndex, BOOL fAscending)

{

if(iIndex < 0)

return E\_FAIL;

// First, get the current header item fmt

HDITEM hdi;

hdi.mask = HDI\_FORMAT;

HRESULT hr = \_wndFuncListHdr.SendMessage(HDM\_GETITEM, iIndex, cast(LPARAM)&hdi) ? S\_OK : E\_FAIL;

if (SUCCEEDED(hr))

{

// Add the image mask and alignment

hdi.mask |= HDI\_IMAGE;

hdi.fmt |= HDF\_IMAGE;

if ((hdi.fmt & HDF\_JUSTIFYMASK) == HDF\_LEFT)

{

hdi.fmt |= HDF\_BITMAP\_ON\_RIGHT;

}

hdi.iImage = fAscending;

hr = \_wndFuncListHdr.SendMessage(HDM\_SETITEM, iIndex, cast(LPARAM)&hdi) ? S\_OK : E\_FAIL;

}

return hr;

}

HRESULT \_RemoveSortIcon(int iIndex)

{

if(iIndex < 0)

return E\_FAIL;

// First, get the current header item fmt

HDITEM hdi;

hdi.mask = HDI\_FORMAT;

HRESULT hr = \_wndFuncListHdr.SendMessage(HDM\_GETITEM, iIndex, cast(LPARAM)&hdi) ? S\_OK : E\_FAIL;

if (SUCCEEDED(hr))

{

// Remove the image mask and alignment

hdi.fmt &= ~HDF\_IMAGE;

if ((hdi.fmt & HDF\_JUSTIFYMASK) == HDF\_LEFT)

{

hdi.fmt &= ~HDF\_BITMAP\_ON\_RIGHT;

}

hr = \_wndFuncListHdr.SendMessage(HDM\_SETITEM, iIndex, cast(LPARAM)&hdi) ? S\_OK : E\_FAIL;

}

return hr;

}

HRESULT \_InsertListViewColumn(ListView lv, const(static\_COLUMNINFO)[] static\_rgColumns, int iIndex, COLUMNID colid,

int cx, bool set = false)

{

LVCOLUMN lvc;

lvc.mask = LVCF\_FMT | LVCF\_TEXT | LVCF\_WIDTH;

lvc.fmt = static\_rgColumns[colid].fmt;

lvc.cx = cx;

HRESULT hr = S\_OK;

string strDisplayName = static\_rgColumns[colid].displayName;

lvc.pszText = \_toUTF16z(strDisplayName);

uint msg = set ? LVM\_SETCOLUMNW : LVM\_INSERTCOLUMNW;

hr = lv.SendMessage(msg, iIndex, cast(LPARAM)&lvc) >= 0 ? S\_OK : E\_FAIL;

if (SUCCEEDED(hr) && lv == \_wndFuncList)

{

HDITEM hdi;

hdi.mask = HDI\_LPARAM;

hdi.lParam = colid;

hr = \_wndFuncListHdr.SendMessage(HDM\_SETITEM, iIndex, cast(LPARAM)&hdi) ? S\_OK : E\_FAIL;

}

return hr;

}

HRESULT \_InsertListViewColumn(int iIndex, COLUMNID colid, int cx, bool set = false)

{

return \_InsertListViewColumn(\_wndFuncList, s\_rgColumns, iIndex, colid, cx, set);

}

HRESULT \_InitializeListColumns(ListView lv, const(COLUMNINFO)[] rgColumns, const(static\_COLUMNINFO)[] static\_rgColumns)

{

lv.SendMessage(LVM\_DELETEALLITEMS);

lv.SendMessage(LVM\_REMOVEALLGROUPS);

bool hasNameColumn = lv.SendMessage(LVM\_GETCOLUMNWIDTH, 0) > 0;

// cannot delete col 0, so keep name

while(lv.SendMessage(LVM\_DELETECOLUMN, 1)) {}

HRESULT hr = S\_OK;

int cColumnsInserted = 0;

for (UINT i = 0; i < rgColumns.length && SUCCEEDED(hr); i++)

{

const(COLUMNINFO)\* ci = &(rgColumns[i]);

if (ci.fVisible)

{

bool set = hasNameColumn ? cColumnsInserted == 0 : false;

hr = \_InsertListViewColumn(lv, static\_rgColumns, cColumnsInserted++, ci.colid, ci.cx, set);

}

}

return hr;

}

HRESULT \_InitializeFuncListColumns()

{

HRESULT hr;

hr = \_InitializeListColumns(\_wndFuncList, \_rgColumns, s\_rgColumns);

hr |= \_InitializeListColumns(\_wndFanInList, default\_FanColumns, s\_rgFanInColumns);

hr |= \_InitializeListColumns(\_wndFanOutList, default\_FanColumns, s\_rgFanOutColumns);

return hr;

}

HRESULT \_InitializeFuncList()

{

\_wndFuncList.SendMessage(LVM\_SETEXTENDEDLISTVIEWSTYLE,

LVS\_EX\_FULLROWSELECT | LVS\_EX\_DOUBLEBUFFER | LVS\_EX\_LABELTIP,

LVS\_EX\_FULLROWSELECT | LVS\_EX\_DOUBLEBUFFER | LVS\_EX\_LABELTIP);

HIMAGELIST himl;

HRESULT hr = \_CreateSortImageList(himl);

if (SUCCEEDED(hr))

{

\_wndFuncListHdr.SendMessage(HDM\_SETIMAGELIST, HDMIL\_PRIVATE, cast(LPARAM)himl);

\_InitializeFuncListColumns();

if (SUCCEEDED(hr))

{

hr = \_AddSortIcon(\_ListViewIndexFromColumnID(\_iqp.colidSort), \_iqp.fSortAscending);

if (SUCCEEDED(hr))

{

\_RefreshFileList();

}

}

}

return hr;

}

// Special icon dimensions for the toolbar images

enum int c\_cxToolbarIcon = 16;

enum int c\_cyToolbarIcon = 15;

HRESULT \_CreateToolbarImageList(out HIMAGELIST phiml)

{

// Create an image list for the sort direction indicators

int icons = IDR\_LAST - IDR\_FIRST + 1;

HIMAGELIST himl = ImageList\_Create(c\_cxToolbarIcon, c\_cyToolbarIcon, ILC\_COLORDDB | ILC\_MASK, icons, 1);

HRESULT hr = himl ? S\_OK : E\_OUTOFMEMORY;

if (SUCCEEDED(hr))

{

// icons have image index IDR\_XXX - IDR\_FIRST

for (int i = IDR\_FIRST; i <= IDR\_LAST && SUCCEEDED(hr); i++)

{

HICON hicn = cast(HICON)LoadImage(getInstance(), MAKEINTRESOURCE(i),

IMAGE\_ICON, c\_cxToolbarIcon, c\_cyToolbarIcon, LR\_DEFAULTCOLOR | LR\_SHARED);

hr = hicn ? S\_OK : HResultFromLastError();

if (SUCCEEDED(hr))

{

hr = ImageList\_ReplaceIcon(himl, -1, hicn) != -1 ? S\_OK : E\_FAIL;

}

}

if (SUCCEEDED(hr))

{

phiml = himl;

himl = null;

}

if (himl)

{

ImageList\_Destroy(himl);

}

}

return hr;

}

HRESULT \_InitializeToolbar()

{

HRESULT hr = \_CreateToolbarImageList(\_himlToolbar);

if (SUCCEEDED(hr))

{

int style = CCS\_NODIVIDER | TBSTYLE\_FLAT | TBSTYLE\_TOOLTIPS | CCS\_NORESIZE;

//style |= (kToolBarAtTop ? CCS\_TOP : CCS\_BOTTOM);

\_wndToolbar = new ToolBar(\_wndBack, style, TBSTYLE\_EX\_DOUBLEBUFFER, IDC\_TOOLBAR);

hr = \_wndToolbar.hwnd ? S\_OK : E\_FAIL;

if (SUCCEEDED(hr))

{

\_wndToolbar.SendMessage(TB\_SETIMAGELIST, 0, cast(LPARAM)\_himlToolbar);

TBBUTTON initButton(int id, ubyte style)

{

return TBBUTTON(id < 0 ? IDR\_LAST - IDR\_FIRST + 1 : id - IDR\_FIRST,

id, TBSTATE\_ENABLED, style, [0,0], 0, 0);

}

static const TBBUTTON[] s\_tbb = [

initButton(IDR\_ALTERNATEROWCOLOR, BTNS\_CHECK),

initButton(IDR\_CLOSEONRETURN, BTNS\_CHECK),

initButton(IDR\_FULLDECO, BTNS\_CHECK),

initButton(IDR\_FANINOUT, BTNS\_CHECK),

initButton(-1, BTNS\_SEP),

initButton(IDR\_SETTRACE, BTNS\_BUTTON),

initButton(IDR\_REMOVETRACE, BTNS\_BUTTON),

initButton(IDR\_REFRESH, BTNS\_BUTTON),

];

hr = \_wndToolbar.SendMessage(TB\_ADDBUTTONS, s\_tbb.length, cast(LPARAM)s\_tbb.ptr) ? S\_OK : E\_FAIL;

if (SUCCEEDED(hr))

{

hr = \_InitializeSwitches();

}

}

}

return hr;

}

HRESULT \_InitializeSwitches()

{

// Set the initial state of the buttons

HRESULT hr = S\_OK;

\_wndToolbar.EnableCheckButton(IDR\_ALTERNATEROWCOLOR, true, \_fAlternateRowColor != 0);

\_wndToolbar.EnableCheckButton(IDR\_CLOSEONRETURN, true, \_closeOnReturn != 0);

//\_wndToolbar.EnableCheckButton(IDR\_GROUPBYKIND, true, \_iqp.colidGroup == COLUMNID.KIND);

\_wndToolbar.EnableCheckButton(IDR\_FANINOUT, true, \_fShowFanInOut != 0);

\_wndToolbar.EnableCheckButton(IDR\_FULLDECO, true, \_fFullDecoration != 0);

return hr;

}

extern(Windows) LRESULT \_HdrWndProc(HWND hwnd, UINT uiMsg, WPARAM wParam, LPARAM lParam)

{

LRESULT lRet = 0;

BOOL fHandled = FALSE;

switch (uiMsg)

{

case WM\_DESTROY:

RemoveWindowSubclass(hwnd, &s\_HdrWndProc, ID\_SUBCLASS\_HDR);

break;

case HDM\_SETIMAGELIST:

if (wParam == HDMIL\_PRIVATE)

{

wParam = 0;

}

else

{

fHandled = TRUE;

}

break;

default:

break;

}

if (!fHandled)

{

lRet = DefSubclassProc(hwnd, uiMsg, wParam, lParam);

}

return lRet;

}

static extern(Windows) LRESULT s\_HdrWndProc(HWND hWnd, UINT uiMsg, WPARAM wParam, LPARAM lParam, UINT\_PTR uIdSubclass, DWORD\_PTR dwRefData)

{

if(ProfilePane pfsec = cast(ProfilePane)cast(void\*)dwRefData)

return pfsec.\_HdrWndProc(hWnd, uiMsg, wParam, lParam);

return DefSubclassProc(hWnd, uiMsg, wParam, lParam);

}

LRESULT \_OnInitDialog(UINT uiMsg, WPARAM wParam, LPARAM lParam, ref BOOL fHandled)

{

if(\_wndFileWheel)

return S\_OK;

updateEnvironmentFont();

if(!mDlgFont)

mDlgFont = newDialogFont();

if (SUCCEEDED(\_InitializeViewState()))

{

\_wndFileWheel = new Text(\_wndBack, "", IDC\_FILEWHEEL);

int top = kToolBarAtTop ? kToolBarHeight : 1;

\_wndFileWheel.setRect(kBackMargin, top + kBackMargin, 185, 16);

\_wndFuncList = new ListView(\_wndBack, LVS\_REPORT | LVS\_SINGLESEL | LVS\_SHOWSELALWAYS | LVS\_ALIGNLEFT | LVS\_SHAREIMAGELISTS | WS\_BORDER | WS\_TABSTOP,

0, IDC\_FILELIST);

\_wndFuncList.setRect(kBackMargin, top + kBackMargin + 20, 185, 78);

HWND hdrHwnd = cast(HWND)\_wndFuncList.SendMessage(LVM\_GETHEADER);

if(hdrHwnd)

{

\_wndFuncListHdr = new Window(hdrHwnd);

// HACK: This header control is created by the listview. When listview handles LVM\_SETIMAGELIST with

// LVSIL\_SMALL it also forwards the message to the header control. The subclass proc will intercept those

// messages and prevent resetting the imagelist

SetWindowSubclass(\_wndFuncListHdr.hwnd, &s\_HdrWndProc, ID\_SUBCLASS\_HDR, cast(DWORD\_PTR)cast(void\*)this);

//\_wndFuncListHdr.SetDlgCtrlID(IDC\_FILELISTHDR);

}

\_wndFanInList = new ListView(\_wndBack, LVS\_REPORT | LVS\_SINGLESEL | LVS\_SHOWSELALWAYS | LVS\_ALIGNLEFT | LVS\_SHAREIMAGELISTS | WS\_BORDER | WS\_TABSTOP,

0, IDC\_FANINLIST);

\_wndFanInList.setRect(kBackMargin, top + 20 + 78, 185, 40);

\_wndFanOutList = new ListView(\_wndBack, LVS\_REPORT | LVS\_SINGLESEL | LVS\_SHOWSELALWAYS | LVS\_ALIGNLEFT | LVS\_SHAREIMAGELISTS | WS\_BORDER | WS\_TABSTOP,

0, IDC\_FANOUTLIST);

\_wndFanOutList.setRect(kBackMargin, top + 20 + 78 + 40, 185, 40);

\_InitializeFuncList();

\_wndFanInList.SendMessage(LVM\_SETEXTENDEDLISTVIEWSTYLE,

LVS\_EX\_FULLROWSELECT | LVS\_EX\_DOUBLEBUFFER | LVS\_EX\_LABELTIP,

LVS\_EX\_FULLROWSELECT | LVS\_EX\_DOUBLEBUFFER | LVS\_EX\_LABELTIP);

\_wndFanOutList.SendMessage(LVM\_SETEXTENDEDLISTVIEWSTYLE,

LVS\_EX\_FULLROWSELECT | LVS\_EX\_DOUBLEBUFFER | LVS\_EX\_LABELTIP,

LVS\_EX\_FULLROWSELECT | LVS\_EX\_DOUBLEBUFFER | LVS\_EX\_LABELTIP);

\_InitializeToolbar();

}

//return CComCompositeControl<CFlatSolutionExplorer>::OnInitDialog(uiMsg, wParam, lParam, fHandled);

return S\_OK;

}

LRESULT \_OnCalcSize(UINT uiMsg, WPARAM wParam, LPARAM lParam, ref BOOL fHandled)

{

//                \_CheckSize();

return 0;

}

void \_CheckSize()

{

RECT r, br;

\_wndParent.GetClientRect(&r);

\_wndBack.GetClientRect(&br);

if(br.right - br.left != r.right - r.left - 2\*kPaneMargin ||

br.bottom - br.top != r.bottom - r.top - 2\*kPaneMargin)

\_wndBack.setRect(kPaneMargin, kPaneMargin,

r.right - r.left - 2\*kPaneMargin, r.bottom - r.top - 2\*kPaneMargin);

}

LRESULT \_OnSize(UINT uiMsg, WPARAM wParam, LPARAM lParam, ref BOOL fHandled)

{

int cx = LOWORD(lParam);

int cy = HIWORD(lParam);

return ResizeControls(cx, cy);

}

LRESULT ResizeControls(int cx, int cy)

{

// Adjust child control sizes

// - File Wheel stretches to fit horizontally but size is vertically fixed

// - File List stretches to fit horizontally and vertically but the topleft coordinate is fixed

// - Toolbar autosizes along the bottom

\_wndToolbar.setRect(kBackMargin, kBackMargin, cx - 2 \* kBackMargin, kToolBarHeight);

int hTool = (kToolBarAtTop ? 0 : kToolBarHeight);

int h = cy - hTool - 2 \* kBackMargin;

int hFan = \_fShowFanInOut ? h / 4 : 0;

int hFunc = h - 2 \* hFan;

RECT rcFileWheel;

if (\_wndFileWheel.GetWindowRect(&rcFileWheel))

{

\_wndBack.ScreenToClient(&rcFileWheel);

rcFileWheel.right = cx - kBackMargin;

\_wndFileWheel.SetWindowPos(null, &rcFileWheel, SWP\_NOMOVE | SWP\_NOZORDER | SWP\_NOACTIVATE);

RECT rcFileList;

if (\_wndFuncList.GetWindowRect(&rcFileList))

{

\_wndBack.ScreenToClient(&rcFileList);

rcFileList.right = cx - kBackMargin;

rcFileList.bottom = hFunc + kBackMargin;

\_wndFuncList.SetWindowPos(null, &rcFileList, SWP\_NOMOVE | SWP\_NOZORDER | SWP\_NOACTIVATE);

rcFileList.top = rcFileList.bottom;

rcFileList.bottom += hFan;

if(\_wndFanInList)

\_wndFanInList.SetWindowPos(null, &rcFileList, SWP\_NOZORDER | SWP\_NOACTIVATE);

rcFileList.top = rcFileList.bottom;

rcFileList.bottom += hFan;

if(\_wndFanOutList)

\_wndFanOutList.SetWindowPos(null, &rcFileList, SWP\_NOZORDER | SWP\_NOACTIVATE);

}

}

return 0;

}

void RearrangeControls()

{

RECT rcBack;

if (\_wndBack.GetWindowRect(&rcBack))

ResizeControls(rcBack.right - rcBack.left, rcBack.bottom - rcBack.top);

}

LRESULT \_OnSetFocus(UINT uiMsg, WPARAM wParam, LPARAM lParam, ref BOOL fHandled)

{

// Skip the CComCompositeControl handling

// CComControl<CFlatSolutionExplorer, CAxDialogImpl<CFlatSolutionExplorer>>::OnSetFocus(uiMsg, wParam, lParam, fHandled);

if(\_wndFileWheel)

{

\_wndFileWheel.SetFocus();

\_wndFileWheel.SendMessage(EM\_SETSEL, 0, cast(LPARAM)-1);

fHandled = TRUE;

}

return 0;

}

LRESULT \_OnKeyDown(UINT uiMsg, WPARAM wParam, LPARAM lParam, ref BOOL fHandled)

{

//HWND hwndFocus = .GetFocus();

//UINT cItems = cast(UINT)\_wndFuncList.SendMessage(LVM\_GETITEMCOUNT);

//if (cItems && hwndFocus == \_wndFileWheel.hwnd)

{

UINT vKey = LOWORD(wParam);

switch(vKey)

{

case VK\_UP:

case VK\_DOWN:

case VK\_PRIOR:

case VK\_NEXT:

fHandled = TRUE;

return \_wndFuncList.SendMessage(uiMsg, wParam, lParam);

// \_MoveSelection(vKey == VK\_DOWN);

case VK\_RETURN:

case VK\_EXECUTE:

return \_OnOpenSelectedItem(0, 0, null, fHandled);

case VK\_ESCAPE:

if(\_closeOnReturn)

sWindowFrame.Hide();

break;

default:

break;

}

}

return 0;

}

HRESULT \_ToggleColumnVisibility(COLUMNID colid)

{

HRESULT hr = E\_FAIL;

COLUMNINFO \*pci = \_ColumnInfoFromColumnID(colid);

BOOL fVisible = !pci.fVisible;

if (fVisible)

{

int iIndex = 0;

BOOL fDone = FALSE;

for (size\_t i = 0; i < \_rgColumns.length && !fDone; i++)

{

COLUMNINFO \*ci = &(\_rgColumns[i]);

if (ci.colid == colid)

{

fDone = TRUE;

}

else if (ci.fVisible)

{

iIndex++;

}

}

hr = \_InsertListViewColumn(iIndex, colid, pci.cx);

if (SUCCEEDED(hr))

{

pci.fVisible = TRUE;

}

}

else

{

int iCol = \_ListViewIndexFromColumnID(colid);

hr = \_wndFuncList.SendMessage(LVM\_DELETECOLUMN, iCol) ? S\_OK : E\_FAIL;

if (SUCCEEDED(hr))

{

pci.fVisible = fVisible;

if (colid == \_iqp.colidSort)

{

hr = \_SetSortColumn(COLUMNID.NAME, 0);

}

}

}

if (SUCCEEDED(hr))

{

\_WriteColumnInfoToRegistry();

}

return hr;

}

HRESULT \_ChooseColumns(POINT pt)

{

HMENU hmnu = CreatePopupMenu();

HRESULT hr = hmnu ? S\_OK : HResultFromLastError();

if (SUCCEEDED(hr))

{

MENUITEMINFO mii;

mii.cbSize = mii.sizeof;

mii.fMask = MIIM\_FTYPE | MIIM\_ID | MIIM\_STATE | MIIM\_STRING;

mii.fType = MFT\_STRING;

// Don't include the first column (COLUMNID.NAME) in the list

for (size\_t i = COLUMNID.NAME + 1; i < \_rgColumns.length && SUCCEEDED(hr); i++)

{

COLUMNINFO \*ci = &(\_rgColumns[i]);

string strDisplayName = s\_rgColumns[ci.colid].displayName;

mii.fState = MFS\_ENABLED;

if (ci.fVisible)

{

mii.fState |= MFS\_CHECKED;

}

mii.wID = ci.colid + IDM\_COLUMNLISTBASE;

mii.dwTypeData = \_toUTF16z(strDisplayName);

if(!InsertMenuItem(hmnu, cast(UINT)i-1, TRUE, &mii))

hr = HResultFromLastError();

}

if (SUCCEEDED(hr))

{

UINT uiCmd = TrackPopupMenuEx(hmnu, TPM\_RETURNCMD | TPM\_NONOTIFY | TPM\_HORIZONTAL | TPM\_TOPALIGN | TPM\_LEFTALIGN, pt.x, pt.y, \_wndBack.hwnd, null);

if (uiCmd)

{

hr = \_ToggleColumnVisibility(cast(COLUMNID)(uiCmd - IDM\_COLUMNLISTBASE));

}

}

DestroyMenu(hmnu);

}

return hr;

}

LRESULT \_OnContextMenu(UINT uiMsg, WPARAM wParam, LPARAM lParam, ref BOOL fHandled)

{

fHandled = FALSE;

HWND hwndContextMenu = cast(HWND)wParam;

// I think the listview is doing the wrong thing with WM\_CONTEXTMENU and using its own HWND even if

// the WM\_CONTEXTMENU originated in the header. Just double check the coordinates to be sure

if (hwndContextMenu == \_wndFuncList.hwnd)

{

RECT rcHdr;

if (\_wndFuncListHdr.GetWindowRect(&rcHdr))

{

POINT pt;

pt.x = GET\_X\_LPARAM(lParam);

pt.y = GET\_Y\_LPARAM(lParam);

if (PtInRect(&rcHdr, pt))

{

fHandled = TRUE;

\_ChooseColumns(pt);

}

}

}

return 0;

}

LRESULT \_OnDestroy(UINT uiMsg, WPARAM wParam, LPARAM lParam, ref BOOL fHandled)

{

if (\_himlToolbar)

{

\_wndToolbar.SendMessage(TB\_SETIMAGELIST, 0, cast(LPARAM)null);

ImageList\_Destroy(\_himlToolbar);

\_himlToolbar = null;

}

fHandled = TRUE;

// return CComCompositeControl<CFlatSolutionExplorer>::OnDestroy(uiMsg, wParam, lParam, fHandled);

return 0;

}

HRESULT \_OpenProfileItem(string pszPath, int line)

{

HRESULT hr = S\_OK;

version(all)

{

hr = OpenFileInSolution(pszPath, line);

}

else

{

if(dte80.DTE2 dte = GetDTE())

{

scope(exit) release(dte);

ComPtr!(dte80.ItemOperations) spvsItemOperations;

hr = dte.ItemOperations(&spvsItemOperations.ptr);

if (SUCCEEDED(hr))

{

ComPtr!(dte80a.Window) spvsWnd;

hr = spvsItemOperations.OpenFile(\_toUTF16z(pszPath), null, &spvsWnd.ptr);

}

}

}

if(hr == S\_OK && \_closeOnReturn)

sWindowFrame.Hide();

return hr;

}

LRESULT \_OnOpenSelectedItem(WORD wNotifyCode, WORD wID, HWND hwndCtl, ref BOOL fHandled)

{

int iSel = \_wndFuncList.SendMessage(LVM\_GETNEXTITEM, cast(WPARAM)-1, LVNI\_SELECTED);

if (iSel != -1)

{

\_OpenProfileItem(iSel);

}

else

{

\_OpenProfileItem(\_wndFileWheel.GetWindowText(), -1);

}

fHandled = TRUE;

return 0;

}

LRESULT \_OnFileWheelChanged(WORD wNotifyCode, WORD wID, HWND hwndCtl, ref BOOL fHandled)

{

fHandled = TRUE;

\_RefreshFileList();

return 0;

}

HRESULT \_SetGroupColumn(COLUMNID colid)

{

\_iqp.colidGroup = colid;

\_WriteViewOptionToRegistry("GroupColumn"w, \_iqp.colidGroup);

return \_RefreshFileList();

}

int \_ListViewIndexFromColumnID(COLUMNID colid)

{

int iCol = -1;

int cCols = \_wndFuncListHdr.SendMessage(HDM\_GETITEMCOUNT);

for (int i = 0; i < cCols && iCol == -1; i++)

{

HDITEM hdi;

hdi.mask = HDI\_LPARAM;

if (\_wndFuncListHdr.SendMessage(HDM\_GETITEM, i, cast(LPARAM)&hdi) && hdi.lParam == colid)

{

iCol = i;

}

}

return iCol;

}

COLUMNINFO \*\_ColumnInfoFromColumnID(COLUMNID colid)

{

COLUMNINFO \*pci = null;

for (size\_t iCol = 0; iCol < \_rgColumns.length && pci is null; iCol++)

{

COLUMNINFO \*ci = &(\_rgColumns[iCol]);

if (ci.colid == colid)

{

pci = ci;

}

}

return pci;

}

LRESULT \_OnCheckBtnClicked(WORD wNotifyCode, WORD wID, HWND hwndCtl, ref BOOL fHandled)

{

TBBUTTONINFO tbbi;

tbbi.cbSize = tbbi.sizeof;

tbbi.dwMask = TBIF\_STATE;

if (\_wndToolbar.SendMessage(TB\_GETBUTTONINFO, wID, cast(LPARAM)&tbbi) != -1)

{

bool checked = !!(tbbi.fsState & TBSTATE\_CHECKED);

switch(wID)

{

case IDR\_ALTERNATEROWCOLOR:

\_fAlternateRowColor = checked;

\_WriteViewOptionToRegistry("AlternateRowColor"w, \_fAlternateRowColor);

\_wndFuncList.InvalidateRect(null, FALSE);

break;

case IDR\_CLOSEONRETURN:

\_closeOnReturn = checked;

\_WriteViewOptionToRegistry("CloseOnReturn"w, \_closeOnReturn);

break;

case IDR\_FANINOUT:

\_fShowFanInOut = checked;

\_WriteViewOptionToRegistry("ShowFanInOut"w, \_fShowFanInOut);

RearrangeControls();

break;

case IDR\_REFRESH:

\_RefreshFileList();

break;

case IDR\_SETTRACE:

if(Config cfg = getCurrentStartupConfig())

{

scope(exit) release(cfg);

string workdir = cfg.GetProjectOptions().replaceEnvironment(cfg.GetProjectOptions().debugworkingdir, cfg);

if(!isAbsolute(workdir))

workdir = cfg.GetProjectDir() ~ "\\" ~ workdir;

string tracelog = workdir ~ "trace.log";

\_wndFileWheel.SetWindowText(tracelog);

\_RefreshFileList();

}

break;

case IDR\_REMOVETRACE:

string fname = \_wndFileWheel.GetWindowText();

if(std.file.exists(fname))

std.file.remove(fname);

\_RefreshFileList();

break;

case IDR\_FULLDECO:

\_fFullDecoration = checked;

\_WriteViewOptionToRegistry("FullDecoration"w, \_fFullDecoration);

\_RefreshFileList();

break;

/+

case IDR\_GROUPBYKIND:

\_SetGroupColumn(checked ? COLUMNID.KIND : COLUMNID.NONE);

break;

+/

default:

return 1;

}

}

fHandled = TRUE;

return 0;

}

////////////////////////////////////////////////////////////////////////

COLUMNID \_ColumnIDFromListViewIndex(int iIndex)

{

COLUMNID colid = COLUMNID.NONE;

HDITEM hdi;

hdi.mask = HDI\_LPARAM;

if (\_wndFuncListHdr.SendMessage(HDM\_GETITEM, iIndex, cast(LPARAM)&hdi))

{

colid = cast(COLUMNID)hdi.lParam;

}

return colid;

}

////////////////////////////////////////////////////////////////////////

LRESULT \_OnFileListGetDispInfo(int idCtrl, in NMHDR \*pnmh, ref BOOL fHandled)

{

NMLVDISPINFO \*pnmlvdi = cast(NMLVDISPINFO \*)pnmh;

if (pnmlvdi.item.mask & LVIF\_TEXT)

{

LVITEM lvi;

lvi.mask = LVIF\_PARAM;

lvi.iItem = pnmlvdi.item.iItem;

if (\_wndFuncList.SendItemMessage(LVM\_GETITEM, lvi))

{

pnmlvdi.item.mask |= LVIF\_DI\_SETITEM;

ProfileItem psiWeak = cast(ProfileItem)cast(void\*)lvi.lParam;

string txt;

switch (\_ColumnIDFromListViewIndex(pnmlvdi.item.iSubItem))

{

case COLUMNID.NAME:

txt = \_demangle(psiWeak.GetName(), \_fFullDecoration != 0);

break;

case COLUMNID.CALLS:

long cb = psiWeak.GetCalls();

txt = to!string(cb);

break;

case COLUMNID.TREETIME:

long cb = psiWeak.GetTreeTime();

cb = cast(long) (cb \* 1000000.0 / \_spsii.mTicksPerSec);

txt = to!string(cb);

break;

case COLUMNID.FUNCTIME:

long cb = psiWeak.GetFuncTime();

cb = cast(long) (cb \* 1000000.0 / \_spsii.mTicksPerSec);

txt = to!string(cb);

break;

case COLUMNID.CALLTIME:

long cb = psiWeak.GetCallTime();

cb = cast(long) (cb \* 1000000.0 / \_spsii.mTicksPerSec);

txt = to!string(cb);

break;

default:

break;

}

wstring wtxt = toUTF16(txt) ~ '\000';

int cnt = min(wtxt.length, pnmlvdi.item.cchTextMax);

pnmlvdi.item.pszText[0..cnt] = wtxt.ptr[0..cnt];

}

}

fHandled = TRUE;

return 0;

}

void \_ReinitViewState(bool refresh)

{

\_WriteViewStateToRegistry();

\_RemoveSortIcon(\_ListViewIndexFromColumnID(\_iqp.colidSort));

\_InitializeViewState();

\_InitializeSwitches();

\_AddSortIcon(\_ListViewIndexFromColumnID(\_iqp.colidSort), \_iqp.fSortAscending);

\_InitializeFuncListColumns();

\_RefreshFileList();

}

RegKey \_GetCurrentRegKey(bool write)

{

GlobalOptions opt = Package.GetGlobalOptions();

opt.getRegistryRoot();

wstring regPath = opt.regUserRoot ~ regPathToolsOptions ~ "\\ProfileSymbolWindow"w;

return new RegKey(opt.hUserKey, regPath, write);

}

HRESULT \_InitializeViewState()

{

HRESULT hr = S\_OK;

try

{

scope RegKey keyWinOpts = \_GetCurrentRegKey(false);

if(keyWinOpts.GetDWORD("ColumnInfoVersion"w, 0) == 1)

{

void[] data = keyWinOpts.GetBinary("ColumnInfo"w);

if(data !is null)

\_rgColumns = cast(COLUMNINFO[])data;

}

\_iqp.colidSort = cast(COLUMNID) keyWinOpts.GetDWORD("SortColumn"w, \_iqp.colidSort);

\_iqp.colidGroup = cast(COLUMNID) keyWinOpts.GetDWORD("GroupColumn"w, \_iqp.colidGroup);

\_fAlternateRowColor = keyWinOpts.GetDWORD("AlternateRowColor"w, \_fAlternateRowColor) != 0;

\_closeOnReturn = keyWinOpts.GetDWORD("closeOnReturn"w, \_closeOnReturn) != 0;

\_fShowFanInOut = keyWinOpts.GetDWORD("ShowFanInOut"w, \_fShowFanInOut) != 0;

\_fFullDecoration = keyWinOpts.GetDWORD("FullDecoration"w, \_fFullDecoration) != 0;

}

catch(Exception e)

{

// ok to fail, defaults still work

}

return hr;

}

HRESULT \_WriteViewStateToRegistry()

{

\_WriteColumnInfoToRegistry();

HRESULT hr = S\_OK;

try

{

scope RegKey keyWinOpts = \_GetCurrentRegKey(true);

keyWinOpts.Set("SortColumn"w, \_iqp.colidSort);

keyWinOpts.Set("GroupColumn"w, \_iqp.colidGroup);

keyWinOpts.Set("SortAscending"w, \_iqp.fSortAscending);

keyWinOpts.Set("AlternateRowColor"w, \_fAlternateRowColor);

keyWinOpts.Set("closeOnReturn"w, \_closeOnReturn);

}

catch(Exception e)

{

hr = E\_FAIL;

}

return hr;

}

HRESULT \_WriteColumnInfoToRegistry()

{

HRESULT hr = S\_OK;

for(int i = 0; i < \_rgColumns.length; i++)

\_rgColumns[i].cx = \_wndFuncList.SendMessage(LVM\_GETCOLUMNWIDTH, \_ListViewIndexFromColumnID(\_rgColumns[i].colid));

try

{

scope RegKey keyWinOpts = \_GetCurrentRegKey(true);

keyWinOpts.Set("ColumnInfoVersion"w, kColumnInfoVersion);

keyWinOpts.Set("ColumnInfo"w, \_rgColumns);

}

catch(Exception e)

{

hr = E\_FAIL;

}

return hr;

}

HRESULT \_WriteViewOptionToRegistry(wstring name, DWORD dw)

{

HRESULT hr = S\_OK;

try

{

scope RegKey keyWinOpts = \_GetCurrentRegKey(true);

keyWinOpts.Set(toUTF16(name), dw);

}

catch(Exception e)

{

hr = E\_FAIL;

}

return hr;

}

HRESULT \_WriteSortInfoToRegistry()

{

HRESULT hr = S\_OK;

try

{

scope RegKey keyWinOpts = \_GetCurrentRegKey(true);

keyWinOpts.Set("SortColumn"w, \_iqp.colidSort);

keyWinOpts.Set("SortAscending"w, \_iqp.fSortAscending);

}

catch(Exception e)

{

hr = E\_FAIL;

}

return hr;

}

HRESULT \_SetSortColumn(COLUMNID colid, int iIndex)

{

HRESULT hr = S\_OK;

bool fSortAscending = true;

if (colid == \_iqp.colidSort)

{

fSortAscending = !\_iqp.fSortAscending;

}

else

{

int iIndexCur = \_ListViewIndexFromColumnID(\_iqp.colidSort);

if (iIndexCur != -1) // Current sort column may have been removed from the list view

{

hr = \_RemoveSortIcon(iIndexCur);

}

}

if (SUCCEEDED(hr))

{

hr = \_AddSortIcon(iIndex, fSortAscending);

if (SUCCEEDED(hr))

{

\_iqp.colidSort = colid;

\_iqp.fSortAscending = fSortAscending;

\_WriteSortInfoToRegistry();

hr = \_RefreshFileList();

}

}

return hr;

}

LRESULT \_OnFileListColumnClick(int idCtrl, ref NMHDR \*pnmh, ref BOOL fHandled)

{

NMLISTVIEW \*pnmlv = cast(NMLISTVIEW \*)pnmh;

\_SetSortColumn(\_ColumnIDFromListViewIndex(pnmlv.iSubItem), pnmlv.iSubItem);

fHandled = TRUE;

return 0;

}

LRESULT \_OnFileListDeleteItem(int idCtrl, ref NMHDR \*pnmh, ref BOOL fHandled)

{

NMLISTVIEW \*pnmlv = cast(NMLISTVIEW \*)pnmh;

ProfileItem psi = cast(ProfileItem)cast(void\*)pnmlv.lParam;

// psi.Release();

fHandled = TRUE;

return 0;

}

LRESULT \_OnFileListItemChanged(int idCtrl, ref NMHDR \*pnmh, ref BOOL fHandled)

{

NMLISTVIEW \*pnmlv = cast(NMLISTVIEW \*)pnmh;

if (pnmlv.uNewState & LVIS\_SELECTED)

{

ProfileItem psi = \_lastResultsArray.GetItem(pnmlv.iItem);

RefreshFanInOutList(psi);

\_lastSelectedItem = pnmlv.iItem;

}

fHandled = TRUE;

return 0;

}

LRESULT \_OnFanInOutListDblClick(bool fanin, ref NMHDR \*pnmh, ref BOOL fHandled)

{

NMLISTVIEW \*pnmlv = cast(NMLISTVIEW \*)pnmh;

ProfileItem psi = \_lastResultsArray.GetItem(\_lastSelectedItem);

if(psi)

{

Fan[] fan = fanin ? psi.mFanIn : psi.mFanOut;

if(pnmlv.iItem >= 0 && pnmlv.iItem < fan.length)

{

string func = fan[pnmlv.iItem].func;

int idx = \_lastResultsArray.findFunc(func);

if(idx >= 0)

{

int sel = \_wndFuncList.SendMessage(LVM\_GETNEXTITEM, cast(WPARAM)-1, LVNI\_SELECTED);

\_UpdateSelection(sel, idx);

}

}

}

fHandled = TRUE;

return 0;

}

HRESULT \_OpenProfileItem(int iIndex)

{

ProfileItem psi = \_lastResultsArray.GetItem(iIndex);

if(!psi)

return E\_FAIL;

SearchData sd;

sd.wholeWord = true;

sd.caseSensitive = true;

sd.noDupsOnSameLine = true;

string name = \_demangle(psi.GetName(), false);

if(std.string.indexOf(name, '.') >= 0)

{

sd.findQualifiedName = true;

sd.names ~= name;

}

else

{

if(name == "\_\_Dmain")

sd.names ~= "main";

else if(name.length > 0 && name[0] == '\_')

sd.names ~= name[1..$]; // assume extern "C", cutoff '\_'

else

sd.names ~= name;

}

Definition[] defs = Package.GetLibInfos().findDefinition(sd);

if(defs.length == 0)

{

showStatusBarText("No definition found for '" ~ sd.names[0] ~ "'");

return S\_FALSE;

}

if(defs.length > 1)

{

// TODO: match types to find best candidate?

showStatusBarText("Multiple definitions found for '" ~ sd.names[0] ~ "'");

}

HRESULT hr = S\_FALSE;

for(int i = 0; i < defs.length && hr != S\_OK; i++)

hr = OpenFileInSolution(defs[i].filename, defs[i].line);

if(hr != S\_OK)

showStatusBarText(format("Cannot open %s(%d) for definition of '%s'", defs[0].filename, defs[0].line, sd.names[0]));

return hr;

}

LRESULT \_OnFileListDblClick(int idCtrl, ref NMHDR \*pnmh, ref BOOL fHandled)

{

NMITEMACTIVATE \*pnmitem = cast(NMITEMACTIVATE\*) pnmh;

if (FAILED(\_OpenProfileItem(pnmitem.iItem)))

{

MessageBeep(MB\_ICONHAND);

}

fHandled = TRUE;

return 0;

}

void \_SetAlternateRowColor()

{

COLORREF cr = GetSysColor(COLOR\_HIGHLIGHT);

BYTE r = GetRValue(cr);

BYTE g = GetGValue(cr);

BYTE b = GetBValue(cr);

BYTE rNew = 236;

BYTE gNew = 236;

BYTE bNew = 236;

if (r > g && r > b)

{

rNew = 244;

}

else if (g > r && g > b)

{

gNew = 244;

}

else

{

bNew = 244;

}

\_crAlternate = RGB(rNew, gNew, bNew);

}

LRESULT \_OnFileListCustomDraw(int idCtrl, ref NMHDR \*pnmh, ref BOOL fHandled)

{

LRESULT lRet = CDRF\_DODEFAULT;

NMLVCUSTOMDRAW \*pnmlvcd = cast(NMLVCUSTOMDRAW \*)pnmh;

switch (pnmlvcd.nmcd.dwDrawStage)

{

case CDDS\_PREPAINT:

\_SetAlternateRowColor();

lRet = CDRF\_NOTIFYITEMDRAW;

break;

case CDDS\_ITEMPREPAINT:

{

// Override the colors so that regardless of the focus state, the control appears focused.

// We can't rely on the pnmlvcd.nmcd.uItemState for this because there is a known bug

// with listviews that have the LVS\_EX\_SHOWSELALWAYS style where this bit is set for

// every item

LVITEM lvi;

lvi.mask = LVIF\_STATE;

lvi.iItem = cast(int)pnmlvcd.nmcd.dwItemSpec;

lvi.stateMask = LVIS\_SELECTED;

if (\_wndFuncList.SendItemMessage(LVM\_GETITEM, lvi) && (lvi.state & LVIS\_SELECTED))

{

pnmlvcd.clrText = GetSysColor(COLOR\_HIGHLIGHTTEXT);

pnmlvcd.clrTextBk = GetSysColor(COLOR\_HIGHLIGHT);

pnmlvcd.nmcd.uItemState &= ~CDIS\_SELECTED;

lRet = CDRF\_NEWFONT;

}

else

{

if (\_fAlternateRowColor && !(pnmlvcd.nmcd.dwItemSpec % 2))

{

// TODO: Eventually, it might be nice to build a color based on COLOR\_HIGHLIGHT.

pnmlvcd.clrTextBk = \_crAlternate;

pnmlvcd.nmcd.uItemState &= ~CDIS\_SELECTED;

lRet = CDRF\_NEWFONT;

}

}

break;

}

default:

break;

}

fHandled = TRUE;

return lRet;

}

LRESULT \_OnFileListHdrItemChanged(int idCtrl, ref NMHDR \*pnmh, ref BOOL fHandled)

{

NMHEADER \*pnmhdr = cast(NMHEADER \*)pnmh;

if (pnmhdr.pitem.mask & HDI\_WIDTH)

{

COLUMNID colid = \_ColumnIDFromListViewIndex(pnmhdr.iItem);

COLUMNINFO \*pci = \_ColumnInfoFromColumnID(colid);

pci.cx = pnmhdr.pitem.cxy;

\_WriteColumnInfoToRegistry();

}

fHandled = TRUE;

return 0;

}

LRESULT \_OnToolbarGetInfoTip(int idCtrl, ref NMHDR \*pnmh, ref BOOL fHandled)

{

NMTBGETINFOTIP \*pnmtbgit = cast(NMTBGETINFOTIP \*)pnmh;

string tip;

switch(pnmtbgit.iItem)

{

case IDR\_ALTERNATEROWCOLOR:

tip = "Toggle alternating row background color";

break;

case IDR\_FULLDECO:

tip = "Show full name decoration";

break;

case IDR\_CLOSEONRETURN:

tip = "Close search window when item selected or focus lost";

break;

case IDR\_FANINOUT:

tip = "Show Fan In/Out";

break;

case IDR\_REFRESH:

tip = "Reread trace log to update display";

break;

case IDR\_SETTRACE:

tip = "Set trace log file from current project";

break;

case IDR\_REMOVETRACE:

tip = "Delete current trace.log to reinit profiling";

break;

default:

break;

}

wstring wtip = toUTF16(tip) ~ '\000';

int cnt = min(wtip.length, pnmtbgit.cchTextMax);

pnmtbgit.pszText[0..cnt] = wtip.ptr[0..cnt];

fHandled = TRUE;

return 0;

}

}

class ItemArray

{

ProfileItem[] mItems;

ProfileItemGroup[] mGroups;

void add(ProfileItem item)

{

mItems ~= item;

}

void addByGroup(string grp, ProfileItem item)

{

for(int i = 0; i < mGroups.length; i++)

if(mGroups[i].GetName() == grp)

return mGroups[i].add(item);

auto group = new ProfileItemGroup(grp);

group.add(item);

mGroups ~= group;

}

int GetCount() const { return max(mItems.length, mGroups.length); }

ProfileItemGroup GetGroup(uint idx) const

{

if(idx >= mGroups.length)

return null;

return cast(ProfileItemGroup)mGroups[idx];

}

ProfileItem GetItem(uint idx) const

{

if(idx >= mItems.length)

return null;

return cast(ProfileItem)mItems[idx];

}

int findFunc(string name)

{

foreach(i, psi; mItems)

if(psi.GetName() == name)

return i;

return -1;

}

void sort(COLUMNID id, bool ascending)

{

void doSort(string method)(ref ProfileItem[] items)

{

if(ascending)

std.algorithm.sort!("a." ~ method ~ "() < b." ~ method ~ "()")(items);

else

std.algorithm.sort!("a." ~ method ~ "() > b." ~ method ~ "()")(items);

}

switch(id)

{

case COLUMNID.NAME:

doSort!"GetName"(mItems);

break;

case COLUMNID.CALLS:

doSort!"GetCalls"(mItems);

break;

case COLUMNID.TREETIME:

doSort!"GetTreeTime"(mItems);

break;

case COLUMNID.FUNCTIME:

doSort!"GetFuncTime"(mItems);

break;

case COLUMNID.CALLTIME:

doSort!"GetCallTime"(mItems);

break;

default:

break;

}

foreach(grp; mGroups)

grp.mArray.sort(id, ascending);

}

}

class ProfileItemGroup

{

this(string name)

{

mName = name;

mArray = new ItemArray;

}

void add(ProfileItem item)

{

mArray.add(item);

}

string GetName() const { return mName; }

const(ItemArray) GetItems() const { return mArray; }

ItemArray mArray;

string mName;

}

struct Fan

{

string func;

long calls;

}

class ProfileItem

{

int GetIconIndex() const { return 0; }

string GetName() const { return mName; }

long GetCalls() const { return mCalls; }

long GetTreeTime() const { return mTreeTime; }

long GetFuncTime() const { return mFuncTime; }

long GetCallTime() const { return mCalls ? mFuncTime / mCalls : 0; }

string mName;

long mCalls;

long mTreeTime;

long mFuncTime;

Fan[] mFanIn;

Fan[] mFanOut;

}

class ProfileItemIndex

{

HRESULT Update(string fname, INDEXQUERYPARAMS \*piqp, ItemArray \*ppv)

{

ItemArray array = new ItemArray;

\*ppv = array;

if(!std.file.exists(fname))

return S\_FALSE;

ubyte[] text; // not valid utf8

try

{

ProfileItem curItem;

File file = File(fname, "rb");

char[] buf;

while(file.readln(buf))

{

if(buf[0] == '-')

{

curItem = new ProfileItem;

array.add(curItem);

}

else if(buf[0] == '=')

{

int pos = std.string.indexOf(buf, "Timer Is");

if(pos > 0)

{

char[] txt = buf[pos + 9 .. $];

mTicksPerSec = parse!long(txt);

}

break;

}

else if(curItem)

{

char[] txt = buf;

munch(txt, " \t\n\r");

if(txt.length > 0 && isDigit(txt[0]))

{

long calls;

if(parseLong(txt, calls))

{

char[] id = parseNonSpace(txt);

if(id.length > 0)

{

munch(txt, " \t\n\r");

if(txt.length == 0)

{

Fan fan = Fan(to!string(id), calls);

if(curItem.mName.length)

curItem.mFanOut ~= fan;

else

curItem.mFanIn ~= fan;

}

}

}

}

else if(txt.length > 0)

{

long calls, treeTime, funcTime;

char[] id = parseNonSpace(txt);

if(id.length > 0 &&

parseLong(txt, calls) &&

parseLong(txt, treeTime) &&

parseLong(txt, funcTime))

{

munch(txt, " \t\n\r");

if(txt.length == 0)

{

curItem.mName = to!string(id);

curItem.mCalls = calls;

curItem.mTreeTime = treeTime;

curItem.mFuncTime = funcTime;

}

}

}

}

}

array.sort(piqp.colidSort, piqp.fSortAscending);

return S\_OK;

}

catch(Exception e)

{

return E\_FAIL;

}

}

long mTicksPerSec = 1;

}

// This file is part of Visual D

//

// Visual D integrates the D programming language into Visual Studio

// Copyright (c) 2010 by Rainer Schuetze, All Rights Reserved

//

// Distributed under the Boost Software License, Version 1.0.

// See accompanying file LICENSE\_1\_0.txt or copy at <http://www.boost.org/LICENSE_1_0.txt>

module visuald.propertypage;

import visuald.windows;

import sdk.win32.objbase;

import sdk.vsi.vsshell;

import sdk.vsi.vsshell80;

import visuald.comutil;

import visuald.logutil;

import visuald.dpackage;

import visuald.dproject;

import visuald.dllmain;

import visuald.config;

import visuald.winctrl;

import visuald.hierarchy;

import visuald.hierutil;

import visuald.pkgutil;

import visuald.chiernode;

import stdext.array;

import stdext.path;

import std.array;

import std.string;

import std.conv;

import std.algorithm;

// version = DParserOption;

class PropertyWindow : Window

{

this(Widget parent, uint style, string title, PropertyPage page)

{

mPropertyPage = page;

super(parent, style, title);

}

override int WindowProc(HWND hWnd, uint uMsg, WPARAM wParam, LPARAM lParam)

{

import sdk.win32.commctrl;

switch (uMsg) {

case WM\_SIZE:

mPropertyPage.updateSizes();

break;

case TCN\_SELCHANGING:

case TCN\_SELCHANGE:

// Return FALSE to allow the selection to change.

auto tc = cast(TabControl) this;

return FALSE;

default:

break;

}

return super.WindowProc(hWnd, uMsg, wParam, lParam);

}

PropertyPage mPropertyPage;

}

abstract class PropertyPage : DisposingComObject, IPropertyPage, IVsPropertyPage, IVsPropertyPage2

{

/\*const\*/ int kPageWidth = 370;

/\*const\*/ int kPageHeight = 210;

/\*const\*/ int kMargin = 2;

/\*const\*/ int kLabelWidth = 120;

/\*const\*/ int kTextHeight = 20;

/\*const\*/ int kLineHeight = 23;

/\*const\*/ int kLineSpacing = 2;

/\*const\*/ int kNeededLines = 11;

override HRESULT QueryInterface(in IID\* riid, void\*\* pvObject)

{

if(queryInterface!(IPropertyPage) (this, riid, pvObject))

return S\_OK;

if(queryInterface!(IVsPropertyPage) (this, riid, pvObject))

return S\_OK;

if(queryInterface!(IVsPropertyPage2) (this, riid, pvObject))

return S\_OK;

return super.QueryInterface(riid, pvObject);

}

override void Dispose()

{

mResizableWidgets = mResizableWidgets.init;

mSite = release(mSite);

foreach(obj; mObjects)

release(obj);

mObjects.length = 0;

mDlgFont = deleteDialogFont(mDlgFont);

}

override int SetPageSite(

/\* [in] \*/ IPropertyPageSite pPageSite)

{

mixin(LogCallMix);

mSite = release(mSite);

mSite = addref(pPageSite);

return S\_OK;

}

override int Activate(

/\* [in] \*/ in HWND hWndParent,

/\* [in] \*/ in RECT \*pRect,

/\* [in] \*/ in BOOL bModal)

{

mixin(LogCallMix);

if(mWindow)

return returnError(E\_FAIL);

return \_Activate(new Window(hWndParent), pRect, bModal != 0);

}

int \_Activate(Window win, const(RECT) \*pRect, bool bModal)

{

updateEnvironmentFont();

if(!mDlgFont)

mDlgFont = newDialogFont();

mWindow = win;

mCanvas = new Window(mWindow);

DWORD color = GetSysColor(COLOR\_BTNFACE);

mCanvas.setBackground(color);

// create with desired size to get proper alignment, then resize to parent later

mCanvas.setRect(kMargin, kMargin, kPageWidth - 2 \* kMargin, kPageHeight - 2 \* kMargin);

// avoid closing canvas (but not dialog) if pressing esc in MultiLineEdit controls

//mCanvas.cancelCloseDelegate ~= delegate bool(Widget c) { return true; };

mCanvas.commandDelegate = &OnCommand;

CreateControls();

UpdateControls();

updateSizes();

mEnableUpdateDirty = true;

return S\_OK;

}

extern(D) void OnCommand(Widget w, int cmd)

{

UpdateDirty(true);

}

override int Deactivate()

{

mixin(LogCallMix);

if(mWindow)

{

auto win = mWindow;

mCanvas = null;

mWindow = null;

win.Dispose();

}

return S\_OK;

//return returnError(E\_NOTIMPL);

}

void updateSizes()

{

if (!mWindow || !mCanvas)

return;

RECT r, pr;

mCanvas.GetWindowRect(&r);

mWindow.GetWindowRect(&pr);

int pageWidth = pr.right - pr.left - 2 \* kMargin;

int pageHeight = pr.bottom - pr.top - 2 \* kMargin;

if (r.right - r.left == pageWidth && r.bottom - r.top == pageHeight)

return;

mCanvas.setRect(kMargin, kMargin, pageWidth, pageHeight);

updateResizableWidgets(mCanvas);

}

void updateResizableWidgets(Widget w)

{

if (auto patt = w in mResizableWidgets)

patt.resizeWidget(w);

foreach(c; w.children)

updateResizableWidgets(c);

}

void addResizableWidget(Widget w, Attachment att)

{

AttachData attData = AttachData(att);

attData.initFromWidget(w);

mResizableWidgets[w] = attData;

}

void refreshResizableWidget(Widget w)

{

if (auto att = w in mResizableWidgets)

att.initFromWidget(w);

}

void addTextPath(Text ctrl, string path, string sep)

{

string imp = ctrl.getText();

if(!imp.empty() && !imp.endsWith(sep))

imp ~= sep;

imp ~= quoteFilename(path);

ctrl.setText(imp);

}

void addBrowsePath(Text ctrl, bool dir, string reldir, string sep, string title, string filter = null)

{

string path;

if(dir)

path = browseDirectory(mCanvas.hwnd, title, reldir);

else

path = browseFile(mCanvas.hwnd, title, filter, reldir);

if (!path.empty)

{

if(reldir)

path = makeRelative(path, reldir);

addTextPath(ctrl, path, sep);

}

}

void calcMetric()

{

updateEnvironmentFont();

if(!mDlgFont)

mDlgFont = newDialogFont();

HWND hwnd = GetDesktopWindow();

HDC dc = GetDC(hwnd);

SelectObject(dc, mDlgFont);

TEXTMETRIC tm;

GetTextMetrics(dc, &tm);

ReleaseDC(hwnd, dc);

int fHeight = tm.tmHeight;

int fWidth = tm.tmAveCharWidth;

kPageWidth = fWidth \* 75 + 2 \* kMargin;

kLabelWidth = fWidth \* 22;

mUnindentCheckBox = kLabelWidth;

kLineSpacing = 2;

kTextHeight = fHeight + 4;

kLineHeight = kTextHeight + kLineSpacing + 1;

kPageHeight = kLineHeight \* kNeededLines + 2 \* kMargin;

}

override int GetPageInfo(

/\* [out] \*/ PROPPAGEINFO \*pPageInfo)

{

mixin(LogCallMix);

if(pPageInfo.cb < PROPPAGEINFO.sizeof)

return E\_INVALIDARG;

calcMetric();

pPageInfo.cb = PROPPAGEINFO.sizeof;

pPageInfo.pszTitle = string2OLESTR("Title");

pPageInfo.size = visuald.comutil.SIZE(kPageWidth, kPageHeight);

pPageInfo.pszHelpFile = string2OLESTR("HelpFile");

pPageInfo.pszDocString = string2OLESTR("DocString");

pPageInfo.dwHelpContext = 0;

return S\_OK;

}

override int SetObjects(

/\* [in] \*/ in ULONG cObjects,

/\* [size\_is][in] \*/ IUnknown \*ppUnk)

{

mixin(LogCallMix2);

foreach(obj; mObjects)

release(obj);

mObjects.length = 0;

for(uint i = 0; i < cObjects; i++)

mObjects ~= addref(ppUnk[i]);

if(mWindow)

{

mEnableUpdateDirty = false;

UpdateControls();

mEnableUpdateDirty = true;

}

return S\_OK;

}

override int Show(

/\* [in] \*/ in UINT nCmdShow)

{

logCall("%s.Show(nCmdShow=%s)", this, \_toLog(nCmdShow));

if(mWindow)

mWindow.setVisible(true);

return S\_OK;

//return returnError(E\_NOTIMPL);

}

override int Move(

/\* [in] \*/ in RECT \*pRect)

{

mixin(LogCallMix);

updateSizes();

return S\_OK; //returnError(E\_NOTIMPL);

}

override int Help(

/\* [in] \*/ in wchar\* pszHelpDir)

{

logCall("%s.Help(pszHelpDir=%s)", this, \_toLog(pszHelpDir));

return returnError(E\_NOTIMPL);

}

override int TranslateAccelerator(

/\* [in] \*/ in MSG \*pMsg)

{

mixin(LogCallMix2);

if(mSite)

return mSite.TranslateAccelerator(pMsg);

return returnError(E\_NOTIMPL);

}

// IVsPropertyPage

override int get\_CategoryTitle(

/\* [in] \*/ in UINT iLevel,

/\* [retval][out] \*/ BSTR \*pbstrCategory)

{

logCall("%s.get\_CategoryTitle(iLevel=%s, pbstrCategory=%s)", this, \_toLog(iLevel), \_toLog(pbstrCategory));

switch(iLevel)

{

case 0:

if(GetCategoryName().length == 0)

return S\_FALSE;

\*pbstrCategory = allocBSTR(GetCategoryName());

break;

case 1:

return S\_FALSE;

//\*pbstrCategory = allocBSTR("CategoryTitle1");

default:

break;

}

return S\_OK;

}

// IVsPropertyPage2

override int GetProperty(

/\* [in] \*/ in VSPPPID propid,

/\* [out] \*/ VARIANT \*pvar)

{

mixin(LogCallMix);

switch(propid)

{

case VSPPPID\_PAGENAME:

pvar.vt = VT\_BSTR;

pvar.bstrVal = allocBSTR(GetPageName());

return S\_OK;

default:

break;

}

return returnError(DISP\_E\_MEMBERNOTFOUND);

}

override int SetProperty(

/\* [in] \*/ in VSPPPID propid,

/\* [in] \*/ in VARIANT var)

{

mixin(LogCallMix);

return returnError(E\_NOTIMPL);

}

///////////////////////////////////////

void UpdateDirty(bool bDirty)

{

if(mEnableUpdateDirty && mSite)

mSite.OnStatusChange(PROPPAGESTATUS\_DIRTY | PROPPAGESTATUS\_VALIDATE);

}

static int getWidgetWidth(Widget w, int def)

{

RECT pr;

if(w && w.GetWindowRect(&pr))

return pr.right - pr.left;

return def;

}

void AddControl(string label, Widget w)

{

AddControl(label, w, null, 0);

}

void AddControl(string label, Widget w, Button btn)

{

AddControl(label, w, btn, 0);

}

void AddControl(string label, Widget w, short attachY)

{

AddControl(label, w, null, attachY);

}

void AddControl(string label, Widget w, Button btn, short resizeY)

{

int x = kLabelWidth;

auto cb = cast(CheckBox) w;

auto tc = cast(TabControl) w;

auto mt = cast(MultiLineText) w;

//if(cb)

//        cb.cmd = 1; // enable actionDelegate

int lines = 1;

if(mt || tc)

lines = mLinesPerMultiLine;

int pageWidth = getWidgetWidth(w ? w.parent : null, kPageWidth);

if (btn)

pageWidth -= kLineHeight;

int labelWidth = 0;

int margin = tc ? 0 : kMargin;

if(label.length)

{

Label lab = new Label(w ? w.parent : null, label);

int off = ((kLineHeight - kLineSpacing) - 16) / 2;

labelWidth = w ? kLabelWidth : pageWidth - 2\*margin;

lab.setRect(0, mLineY + off, labelWidth, kLineHeight - kLineSpacing);

if(mAttachY > 0)

{

Attachment att = kAttachNone;

att.vdiv = 1000;

att.top = att.bottom = mAttachY;

addResizableWidget(lab, att);

}

}

else if (cb || tc)

{

x -= mUnindentCheckBox;

}

int h = lines \* kLineHeight - kLineSpacing;

if(cast(Text) w && lines == 1)

{

h = kTextHeight;

}

else if(cb)

h -= 2;

else if(tc)

h += tc.getFrameHeight() - kLineHeight;

//else if(cast(ComboBox) w)

// h -= 4;

int yspacing = (lines \* kLineHeight - kLineSpacing - h) / 2;

int y = mLineY + max(0, yspacing);

if(w)

{

w.setRect(x, y, pageWidth - 2\*margin - labelWidth, h);

Attachment att = kAttachLeftRight;

att.vdiv = 1000;

att.top = mAttachY;

att.bottom = cast(short)(mAttachY + resizeY);

addResizableWidget(w, att);

}

if(btn)

{

btn.setRect(pageWidth - kMargin, y, kLineHeight, kLineHeight - kLineSpacing);

Attachment att = kAttachRight;

att.vdiv = 1000;

att.top = att.bottom = mAttachY;

addResizableWidget(btn, att);

}

mLineY += max(h, lines \* kLineHeight);

mAttachY += resizeY;

}

void AddHorizontalLine()

{

auto w = new Label(mCanvas);

w.AddWindowStyle(SS\_ETCHEDFRAME, SS\_TYPEMASK);

w.setRect(0, mLineY + 2, getWidgetWidth(mCanvas, kPageWidth) - 2\*kMargin, 2);

Attachment att = kAttachLeftRight;

att.vdiv = 1000;

att.top = att.bottom = mAttachY;

addResizableWidget(w, att);

mLineY += 6;

}

int changeOption(V)(V val, ref V optval, ref V refval)

{

if(refval == val)

return 0;

optval = val;

return 1;

}

int changeOptionDg(V)(V val, void delegate (V optval) setdg, V refval)

{

if(refval == val)

return 0;

setdg(val);

return 1;

}

abstract void CreateControls();

abstract void UpdateControls();

abstract string GetCategoryName();

abstract string GetPageName();

AttachData[Widget] mResizableWidgets;

HFONT mDlgFont;

IUnknown[] mObjects;

IPropertyPageSite mSite;

Window mWindow;

Window mCanvas;

bool mEnableUpdateDirty;

int mLineY;

short mAttachY = 0; // fraction of 1000

int mLinesPerMultiLine = 4;

int mUnindentCheckBox = 120; //16;

}

///////////////////////////////////////////////////////////////////////////////

class ProjectPropertyPage : PropertyPage, ConfigModifiedListener

{

abstract void SetControls(ProjectOptions options);

abstract int DoApply(ProjectOptions options, ProjectOptions refoptions);

override HRESULT QueryInterface(in IID\* riid, void\*\* pvObject)

{

//if(queryInterface!(ConfigModifiedListener) (this, riid, pvObject))

//        return S\_OK;

return super.QueryInterface(riid, pvObject);

}

override void UpdateControls()

{

if(ProjectOptions options = GetProjectOptions())

SetControls(options);

}

override void Dispose()

{

if(auto cfg = GetConfig())

cfg.RemoveModifiedListener(this);

super.Dispose();

}

override void OnConfigModified()

{

}

override int SetObjects(/\* [in] \*/ in ULONG cObjects,

/\* [size\_is][in] \*/ IUnknown \*ppUnk)

{

if(auto cfg = GetConfig())

cfg.RemoveModifiedListener(this);

int rc = super.SetObjects(cObjects, ppUnk);

if(auto cfg = GetConfig())

cfg.AddModifiedListener(this);

return rc;

}

Config GetConfig()

{

if(mObjects.length > 0)

{

auto config = ComPtr!(Config)(mObjects[0]);

return config;

}

return null;

}

ProjectOptions GetProjectOptions()

{

if(auto cfg = GetConfig())

return cfg.GetProjectOptions();

return null;

}

string GetProjectDir()

{

if(auto cfg = GetConfig())

return cfg.GetProjectDir();

return null;

}

/\*override\*/ int IsPageDirty()

{

mixin(LogCallMix);

if(mWindow)

if(ProjectOptions options = GetProjectOptions())

{

scope ProjectOptions opt = new ProjectOptions(false, false);

return DoApply(opt, options) > 0 ? S\_OK : S\_FALSE;

}

return S\_FALSE;

}

/\*override\*/ int Apply()

{

mixin(LogCallMix);

if(ProjectOptions refoptions = GetProjectOptions())

{

// make a copy, otherwise changes will no longer be detected after the first configuration

auto refopt = clone(refoptions);

for(int i = 0; i < mObjects.length; i++)

{

auto config = ComPtr!(Config)(mObjects[i]);

if(config)

{

DoApply(config.ptr.GetProjectOptions(), refopt);

config.SetDirty();

}

}

return S\_OK;

}

return returnError(E\_FAIL);

}

}

class NodePropertyPage : PropertyPage

{

abstract void SetControls(CFileNode node);

abstract int DoApply(CFileNode node, CFileNode refnode);

override void UpdateControls()

{

if(CFileNode node = GetNode())

SetControls(node);

}

CFileNode GetNode()

{

if(mObjects.length > 0)

{

auto node = ComPtr!(CFileNode)(mObjects[0]);

if(node)

return node;

}

return null;

}

/\*override\*/ int IsPageDirty()

{

mixin(LogCallMix);

if(mWindow)

if(CFileNode node = GetNode())

{

scope CFileNode n = newCom!CFileNode("");

return DoApply(n, node) > 0 ? S\_OK : S\_FALSE;

}

return S\_FALSE;

}

/\*override\*/ int Apply()

{

mixin(LogCallMix);

if(CFileNode refnode = GetNode())

{

for(int i = 0; i < mObjects.length; i++)

{

auto node = ComPtr!(CFileNode)(mObjects[i]);

if(node)

{

DoApply(node, refnode);

if(CProjectNode pn = cast(CProjectNode) node.GetRootNode())

pn.SetProjectFileDirty(true);

}

}

return S\_OK;

}

return returnError(E\_FAIL);

}

}

class GlobalPropertyPage : PropertyPage

{

abstract void SetControls(GlobalOptions options);

abstract int DoApply(GlobalOptions options, GlobalOptions refoptions);

this(GlobalOptions options)

{

mOptions = options;

}

override void UpdateControls()

{

if(GlobalOptions options = GetGlobalOptions())

SetControls(options);

}

GlobalOptions GetGlobalOptions()

{

return mOptions;

}

void SetWindowSize(int x, int y, int w, int h)

{

mixin(LogCallMix);

if(mCanvas)

mCanvas.setRect(x, y, w, h);

}

/\*override\*/ int IsPageDirty()

{

mixin(LogCallMix);

if(mWindow)

if(GlobalOptions options = GetGlobalOptions())

{

scope GlobalOptions opt = new GlobalOptions;

return DoApply(opt, options) > 0 ? S\_OK : S\_FALSE;

}

return S\_FALSE;

}

/\*override\*/ int Apply()

{

mixin(LogCallMix);

if(GlobalOptions options = GetGlobalOptions())

{

DoApply(options, options);

options.saveToRegistry();

return S\_OK;

}

return returnError(E\_FAIL);

}

GlobalOptions mOptions;

}

///////////////////////////////////////////////////////////////////////////////

class CommonPropertyPage : ProjectPropertyPage

{

override string GetCategoryName() { return ""; }

override string GetPageName() { return "General"; }

override void CreateControls()

{

AddControl("Build System", mCbBuildSystem = new ComboBox(mCanvas, [ "Visual D", "dsss", "rebuild" ], false));

mCbBuildSystem.setSelection(0);

mCbBuildSystem.setEnabled(false);

}

override void SetControls(ProjectOptions options)

{

}

override int DoApply(ProjectOptions options, ProjectOptions refoptions)

{

return 0;

}

ComboBox mCbBuildSystem;

}

class GeneralPropertyPage : ProjectPropertyPage

{

override string GetCategoryName() { return ""; }

override string GetPageName() { return "General"; }

\_\_gshared const float[] selectableVersions = [ 1, 2 ];

override void CreateControls()

{

string[] versions;

foreach(ver; selectableVersions)

versions ~= "D" ~ to!(string)(ver);

//versions[$-1] ~= "+";

AddControl("Compiler", mCompiler = new ComboBox(mCanvas, [ "DMD", "GDC", "LDC" ], false));

AddControl("D-Version", mDVersion = new ComboBox(mCanvas, versions, false));

AddControl("Output Type", mCbOutputType = new ComboBox(mCanvas,

[ "Executable", "Library", "DLL" ], false));

AddControl("Subsystem", mCbSubsystem = new ComboBox(mCanvas,

[ "Not set", "Console", "Windows", "Native", "Posix" ], false));

AddControl("Output Path", mOutputPath = new Text(mCanvas));

AddControl("Intermediate Path", mIntermediatePath = new Text(mCanvas));

AddControl("Files to clean", mFilesToClean = new Text(mCanvas));

AddControl("Compilation", mSingleFileComp = new ComboBox(mCanvas,

[ "Combined compile and link", "Single file compilation",

"Separate compile and link", "Compile only (use Post-build command to link)" ], false));

}

override void SetControls(ProjectOptions options)

{

int ver = 0;

while(ver < selectableVersions.length - 1 && selectableVersions[ver+1] <= options.Dversion)

ver++;

mDVersion.setSelection(ver);

mCompiler.setSelection(options.compiler);

mSingleFileComp.setSelection(options.compilationModel);

mCbOutputType.setSelection(options.lib);

mCbSubsystem.setSelection(options.subsystem);

mOutputPath.setText(options.outdir);

mIntermediatePath.setText(options.objdir);

mFilesToClean.setText(options.filesToClean);

}

override int DoApply(ProjectOptions options, ProjectOptions refoptions)

{

float ver = selectableVersions[mDVersion.getSelection()];

int changes = 0;

changes += changeOption(cast(uint) mSingleFileComp.getSelection(), options.compilationModel, refoptions.compilationModel);

changes += changeOption(cast(ubyte) mCbOutputType.getSelection(), options.lib, refoptions.lib);

changes += changeOption(cast(ubyte) mCbSubsystem.getSelection(), options.subsystem, refoptions.subsystem);

changes += changeOption(cast(ubyte) mCompiler.getSelection(), options.compiler, refoptions.compiler);

changes += changeOption(ver, options.Dversion, refoptions.Dversion);

changes += changeOption(mOutputPath.getText(), options.outdir, refoptions.outdir);

changes += changeOption(mIntermediatePath.getText(), options.objdir, refoptions.objdir);

changes += changeOption(mFilesToClean.getText(), options.filesToClean, refoptions.filesToClean);

return changes;

}

ComboBox mCompiler;

ComboBox mSingleFileComp;

ComboBox mCbOutputType;

ComboBox mCbSubsystem;

ComboBox mDVersion;

Text mOutputPath;

Text mIntermediatePath;

Text mFilesToClean;

}

class DebuggingPropertyPage : ProjectPropertyPage

{

override string GetCategoryName() { return ""; }

override string GetPageName() { return "Debugging"; }

enum ID\_DBGCOMMAND = 1020;

enum ID\_DBGDIR = 1021;

extern(D) override void OnCommand(Widget w, int cmd)

{

switch(cmd)

{

case ID\_DBGCOMMAND:

if(auto file = browseFile(mCanvas.hwnd, "Select executable", "Executables\0\*.exe\0All Files\0\*.\*\0"))

mCommand.setText(file);

break;

case ID\_DBGDIR:

if(auto dir = browseDirectory(mCanvas.hwnd, "Select working directory"))

mWorkingDir.setText(dir);

break;

default:

break;

}

super.OnCommand(w, cmd);

}

override void CreateControls()

{

Label lbl;

auto btn = new Button(mCanvas, "...", ID\_DBGCOMMAND);

AddControl("Command", mCommand = new Text(mCanvas), btn);

AddControl("Command Arguments", mArguments = new Text(mCanvas));

btn = new Button(mCanvas, "...", ID\_DBGDIR);

AddControl("Working Directory", mWorkingDir = new Text(mCanvas), btn);

AddControl("", mAttach = new CheckBox(mCanvas, "Attach to running process"));

AddControl("Remote Machine", mRemote = new Text(mCanvas));

AddControl("Debugger", mDebugEngine = new ComboBox(mCanvas, [ "Visual Studio", "Mago", "Visual Studio (x86 Mixed Mode)" ], false));

AddControl("", mStdOutToOutputWindow = new CheckBox(mCanvas, "Redirect stdout to output window"));

AddControl("Run without debugging", lbl = new Label(mCanvas, ""));

AddControl("", mPauseAfterRunning = new CheckBox(mCanvas, "Pause when program finishes"));

lbl.AddWindowExStyle(WS\_EX\_STATICEDGE);

lbl.AddWindowStyle(SS\_ETCHEDFRAME, SS\_TYPEMASK);

int left, top, w, h;

if(lbl.getRect(left, top, w, h))

lbl.setRect(left, top + h / 2 - 1, w, 2);

refreshResizableWidget(lbl);

}

override void UpdateDirty(bool bDirty)

{

super.UpdateDirty(bDirty);

EnableControls();

}

void EnableControls()

{

mStdOutToOutputWindow.setEnabled(mDebugEngine.getSelection() != 1);

}

override void SetControls(ProjectOptions options)

{

mCommand.setText(options.debugtarget);

mArguments.setText(options.debugarguments);

mWorkingDir.setText(options.debugworkingdir);

mAttach.setChecked(options.debugattach);

mRemote.setText(options.debugremote);

mDebugEngine.setSelection(options.debugEngine);

mStdOutToOutputWindow.setChecked(options.debugStdOutToOutputWindow);

mPauseAfterRunning.setChecked(options.pauseAfterRunning);

EnableControls();

}

override int DoApply(ProjectOptions options, ProjectOptions refoptions)

{

int changes = 0;

changes += changeOption(mCommand.getText(), options.debugtarget, refoptions.debugtarget);

changes += changeOption(mArguments.getText(), options.debugarguments, refoptions.debugarguments);

changes += changeOption(mWorkingDir.getText(), options.debugworkingdir, refoptions.debugworkingdir);

changes += changeOption(mAttach.isChecked(), options.debugattach, options.debugattach);

changes += changeOption(mRemote.getText(), options.debugremote, refoptions.debugremote);

changes += changeOption(cast(ubyte)mDebugEngine.getSelection(), options.debugEngine, refoptions.debugEngine);

changes += changeOption(mStdOutToOutputWindow.isChecked(), options.debugStdOutToOutputWindow, options.debugStdOutToOutputWindow);

changes += changeOption(mPauseAfterRunning.isChecked(), options.pauseAfterRunning, options.pauseAfterRunning);

return changes;

}

Text mCommand;

Text mArguments;

Text mWorkingDir;

Text mRemote;

CheckBox mAttach;

ComboBox mDebugEngine;

CheckBox mStdOutToOutputWindow;

CheckBox mPauseAfterRunning;

}

class DmdGeneralPropertyPage : ProjectPropertyPage

{

override string GetCategoryName() { return "Compiler"; }

override string GetPageName() { return "General"; }

enum ID\_IMPORTPATH = 1030;

enum ID\_STRINGIMPORTPATH = 1031;

void addImportDir(Text ctrl, string title)

{

addBrowsePath(ctrl, true, GetProjectDir(), ";", title);

}

extern(D) override void OnCommand(Widget w, int cmd)

{

switch(cmd)

{

case ID\_IMPORTPATH:

addImportDir(mAddImports, "Add import path");

break;

case ID\_STRINGIMPORTPATH:

addImportDir(mStringImports, "Add string import path");

break;

default:

break;

}

super.OnCommand(w, cmd);

}

override void CreateControls()

{

//AddControl("", mUseStandard = new CheckBox(mCanvas, "Use Standard Import Paths"));

auto btn = new Button(mCanvas, "+", ID\_IMPORTPATH);

AddControl("Additional Import Paths", mAddImports = new Text(mCanvas), btn);

btn = new Button(mCanvas, "+", ID\_STRINGIMPORTPATH);

AddControl("String Import Paths", mStringImports = new Text(mCanvas), btn);

AddControl("Version Identifiers", mVersionIdentifiers = new Text(mCanvas));

AddControl("Debug Identifiers", mDebugIdentifiers = new Text(mCanvas));

AddHorizontalLine();

AddControl("", mOtherDMD = new CheckBox(mCanvas, "Use other compiler"));

AddControl("Compiler Path", mCompilerPath = new Text(mCanvas));

AddHorizontalLine();

AddControl("C/C++ Compiler Cmd", mCCCmd = new Text(mCanvas));

AddControl("", mTransOpt = new CheckBox(mCanvas, "Translate D options (debug, optimizations)"));

}

override void UpdateDirty(bool bDirty)

{

super.UpdateDirty(bDirty);

EnableControls();

}

void EnableControls()

{

mCompilerPath.setEnabled(mOtherDMD.isChecked());

}

override void SetControls(ProjectOptions options)

{

//mUseStandard.setChecked(true);

//mUseStandard.setEnabled(false);

mAddImports.setText(options.imppath);

mStringImports.setText(options.fileImppath);

mVersionIdentifiers.setText(options.versionids);

mDebugIdentifiers.setText(options.debugids);

mOtherDMD.setChecked(options.otherDMD);

mCompilerPath.setText(options.program);

mCCCmd.setText(options.cccmd);

mTransOpt.setChecked(options.ccTransOpt);

EnableControls();

}

override int DoApply(ProjectOptions options, ProjectOptions refoptions)

{

int changes = 0;

changes += changeOption(mAddImports.getText(), options.imppath, refoptions.imppath);

changes += changeOption(mStringImports.getText(), options.fileImppath, refoptions.fileImppath);

changes += changeOption(mVersionIdentifiers.getText(), options.versionids, refoptions.versionids);

changes += changeOption(mDebugIdentifiers.getText(), options.debugids, refoptions.debugids);

changes += changeOption(mOtherDMD.isChecked(), options.otherDMD, refoptions.otherDMD);

changes += changeOption(mCompilerPath.getText(), options.program, refoptions.program);

changes += changeOption(mCCCmd.getText(), options.cccmd, refoptions.cccmd);

changes += changeOption(mTransOpt.isChecked(), options.ccTransOpt, refoptions.ccTransOpt);

return changes;

}

//CheckBox mUseStandard;

Text mAddImports;

Text mStringImports;

Text mVersionIdentifiers;

Text mDebugIdentifiers;

CheckBox mOtherDMD;

Text mCompilerPath;

Text mCCCmd;

CheckBox mTransOpt;

}

class DmdDebugPropertyPage : ProjectPropertyPage

{

override string GetCategoryName() { return "Compiler"; }

override string GetPageName() { return "Debug"; }

enum ID\_BROWSECV2PDB = 1010;

extern(D) override void OnCommand(Widget w, int cmd)

{

switch(cmd)

{

case ID\_BROWSECV2PDB:

if(auto file = browseFile(mCanvas.hwnd, "Select cv2pdb executable", "Executables\0\*.exe\0All Files\0\*.\*\0"))

mPathCv2pdb.setText(file);

break;

default:

break;

}

super.OnCommand(w, cmd);

}

override void CreateControls()

{

string[] dbgInfoOpt = [ "None", "Symbolic (suitable for Mago)", "Symbolic (suitable for VS debug engine)", "Symbolic (suitable for selected debug engine)" ];

AddControl("Debug Mode", mDebugMode = new ComboBox(mCanvas, [ "On (enable debug statements, asserts, invariants and constraints)",

"Off (disable asserts, invariants and constraints)",

"Default (enable asserts, invariants and constraints)" ], false));

AddControl("Debug Info", mDebugInfo = new ComboBox(mCanvas, dbgInfoOpt, false));

AddHorizontalLine();

AddControl("", mRunCv2pdb = new CheckBox(mCanvas, "Run cv2pdb to Convert Debug Info"));

auto btn = new Button(mCanvas, "...", ID\_BROWSECV2PDB);

AddControl("Path to cv2pdb", mPathCv2pdb = new Text(mCanvas), btn);

AddControl("", mCv2pdbPre2043 = new CheckBox(mCanvas, "Assume old associative array implementation (before dmd 2.043)"));

AddControl("", mCv2pdbNoDemangle = new CheckBox(mCanvas, "Do not demangle symbols"));

AddControl("", mCv2pdbEnumType = new CheckBox(mCanvas, "Use enumerator types"));

AddControl("More options", mCv2pdbOptions = new Text(mCanvas));

}

override void UpdateDirty(bool bDirty)

{

super.UpdateDirty(bDirty);

EnableControls();

}

void EnableControls()

{

mRunCv2pdb.setEnabled(mCanRunCv2PDB);

bool runcv2pdb = mCanRunCv2PDB && mRunCv2pdb.isChecked();

mPathCv2pdb.setEnabled(runcv2pdb);

mCv2pdbOptions.setEnabled(runcv2pdb);

mCv2pdbEnumType.setEnabled(runcv2pdb);

mCv2pdbPre2043.setEnabled(runcv2pdb);

mCv2pdbNoDemangle.setEnabled(runcv2pdb);

}

override void SetControls(ProjectOptions options)

{

mDebugMode.setSelection(options.release);

mDebugInfo.setSelection(options.symdebug);

mRunCv2pdb.setChecked(options.runCv2pdb);

mPathCv2pdb.setText(options.pathCv2pdb);

mCv2pdbOptions.setText(options.cv2pdbOptions);

mCv2pdbPre2043.setChecked(options.cv2pdbPre2043);

mCv2pdbNoDemangle.setChecked(options.cv2pdbNoDemangle);

mCv2pdbEnumType.setChecked(options.cv2pdbEnumType);

mCanRunCv2PDB = options.compiler != Compiler.DMD || (!options.isX86\_64 && !options.mscoff);

EnableControls();

}

override int DoApply(ProjectOptions options, ProjectOptions refoptions)

{

int changes = 0;

changes += changeOption(cast(ubyte) mDebugMode.getSelection(), options.release, refoptions.release);

changes += changeOption(cast(ubyte) mDebugInfo.getSelection(), options.symdebug, refoptions.symdebug);

changes += changeOption(mRunCv2pdb.isChecked(), options.runCv2pdb, refoptions.runCv2pdb);

changes += changeOption(mPathCv2pdb.getText(), options.pathCv2pdb, refoptions.pathCv2pdb);

changes += changeOption(mCv2pdbOptions.getText(), options.cv2pdbOptions, refoptions.cv2pdbOptions);

changes += changeOption(mCv2pdbPre2043.isChecked(), options.cv2pdbPre2043, refoptions.cv2pdbPre2043);

changes += changeOption(mCv2pdbNoDemangle.isChecked(), options.cv2pdbNoDemangle, refoptions.cv2pdbNoDemangle);

changes += changeOption(mCv2pdbEnumType.isChecked(), options.cv2pdbEnumType, refoptions.cv2pdbEnumType);

return changes;

}

bool mCanRunCv2PDB;

ComboBox mDebugMode;

ComboBox mDebugInfo;

CheckBox mRunCv2pdb;

Text mPathCv2pdb;

CheckBox mCv2pdbPre2043;

CheckBox mCv2pdbNoDemangle;

CheckBox mCv2pdbEnumType;

Text mCv2pdbOptions;

}

class DmdCodeGenPropertyPage : ProjectPropertyPage

{

this()

{

kNeededLines = 12;

}

override string GetCategoryName() { return "Compiler"; }

override string GetPageName() { return "Code Generation"; }

override void CreateControls()

{

mUnindentCheckBox = kLabelWidth;

AddControl("", mProfiling = new CheckBox(mCanvas, "Insert Profiling Hooks"));

AddControl("", mCodeCov = new CheckBox(mCanvas, "Generate Code Coverage"));

AddControl("", mUnitTests = new CheckBox(mCanvas, "Generate Unittest Code"));

AddHorizontalLine();

AddControl("", mOptimizer = new CheckBox(mCanvas, "Run Optimizer"));

AddControl("", mNoboundscheck = new CheckBox(mCanvas, "No Array Bounds Checking"));

AddControl("", mInline = new CheckBox(mCanvas, "Expand Inline Functions"));

AddHorizontalLine();

AddControl("", mNoFloat = new CheckBox(mCanvas, "No Floating Point Support"));

AddControl("", mGenStackFrame = new CheckBox(mCanvas, "Always generate stack frame (DMD 2.056+)"));

AddControl("", mStackStomp = new CheckBox(mCanvas, "Add stack stomp code (DMD 2.062+)"));

AddControl("", mAllInst = new CheckBox(mCanvas, "Generate code for all template instantiations (DMD 2.064+)"));

}

override void SetControls(ProjectOptions options)

{

mProfiling.setChecked(options.trace);

mCodeCov.setChecked(options.cov);

mOptimizer.setChecked(options.optimize);

mNoboundscheck.setChecked(options.noboundscheck);

mUnitTests.setChecked(options.useUnitTests);

mInline.setChecked(options.useInline);

mNoFloat.setChecked(options.nofloat);

mGenStackFrame.setChecked(options.genStackFrame);

mStackStomp.setChecked(options.stackStomp);

mAllInst.setChecked(options.allinst);

mNoboundscheck.setEnabled(options.Dversion > 1);

}

override int DoApply(ProjectOptions options, ProjectOptions refoptions)

{

int changes = 0;

changes += changeOption(mCodeCov.isChecked(), options.cov, refoptions.cov);

changes += changeOption(mProfiling.isChecked(), options.trace, refoptions.trace);

changes += changeOption(mOptimizer.isChecked(), options.optimize, refoptions.optimize);

changes += changeOption(mNoboundscheck.isChecked(), options.noboundscheck, refoptions.noboundscheck);

changes += changeOption(mUnitTests.isChecked(), options.useUnitTests, refoptions.useUnitTests);

changes += changeOption(mInline.isChecked(), options.useInline, refoptions.useInline);

changes += changeOption(mNoFloat.isChecked(), options.nofloat, refoptions.nofloat);

changes += changeOption(mGenStackFrame.isChecked(), options.genStackFrame, refoptions.genStackFrame);

changes += changeOption(mStackStomp.isChecked(), options.stackStomp, refoptions.stackStomp);

changes += changeOption(mAllInst.isChecked(), options.allinst, refoptions.allinst);

return changes;

}

CheckBox mCodeCov;

CheckBox mProfiling;

CheckBox mOptimizer;

CheckBox mNoboundscheck;

CheckBox mUnitTests;

CheckBox mInline;

CheckBox mNoFloat;

CheckBox mGenStackFrame;

CheckBox mStackStomp;

CheckBox mAllInst;

}

class DmdMessagesPropertyPage : ProjectPropertyPage

{

override string GetCategoryName() { return "Compiler"; }

override string GetPageName() { return "Messages"; }

override void CreateControls()

{

mUnindentCheckBox = kLabelWidth;

AddControl("", mWarnings = new CheckBox(mCanvas, "Enable Warnings"));

AddControl("", mInfoWarnings = new CheckBox(mCanvas, "Enable Informational Warnings (DMD 2.041+)"));

AddHorizontalLine();

AddControl("", mUseDeprecated = new CheckBox(mCanvas, "Silently Allow Deprecated Features"));

AddControl("", mErrDeprecated = new CheckBox(mCanvas, "Use of Deprecated Features causes Error (DMD 2.061+)"));

AddHorizontalLine();

AddControl("", mVerbose = new CheckBox(mCanvas, "Verbose Compile"));

AddControl("", mVtls = new CheckBox(mCanvas, "Show TLS Variables"));

AddControl("", mVgc = new CheckBox(mCanvas, "List all gc allocations including hidden ones (DMD 2.066+)"));

AddControl("", mIgnorePragmas = new CheckBox(mCanvas, "Ignore Unsupported Pragmas"));

AddControl("", mCheckProperty = new CheckBox(mCanvas, "Enforce Property Syntax (DMD 2.055+)"));

}

override void SetControls(ProjectOptions options)

{

mWarnings.setChecked(options.warnings);

mInfoWarnings.setChecked(options.infowarnings);

mVerbose.setChecked(options.verbose);

mVtls.setChecked(options.vtls);

mVgc.setChecked(options.vgc);

mUseDeprecated.setChecked(options.useDeprecated);

mErrDeprecated.setChecked(options.errDeprecated);

mIgnorePragmas.setChecked(options.ignoreUnsupportedPragmas);

mCheckProperty.setChecked(options.checkProperty);

mVtls.setEnabled(options.Dversion > 1);

mVgc.setEnabled(options.Dversion > 1);

}

override int DoApply(ProjectOptions options, ProjectOptions refoptions)

{

int changes = 0;

changes += changeOption(mWarnings.isChecked(), options.warnings, refoptions.warnings);

changes += changeOption(mInfoWarnings.isChecked(), options.infowarnings, refoptions.infowarnings);

changes += changeOption(mVerbose.isChecked(), options.verbose, refoptions.verbose);

changes += changeOption(mVtls.isChecked(), options.vtls, refoptions.vtls);

changes += changeOption(mVgc.isChecked(), options.vgc, refoptions.vgc);

changes += changeOption(mUseDeprecated.isChecked(), options.useDeprecated, refoptions.useDeprecated);

changes += changeOption(mErrDeprecated.isChecked(), options.errDeprecated, refoptions.errDeprecated);

changes += changeOption(mIgnorePragmas.isChecked(), options.ignoreUnsupportedPragmas, refoptions.ignoreUnsupportedPragmas);

changes += changeOption(mCheckProperty.isChecked(), options.checkProperty, refoptions.checkProperty);

return changes;

}

CheckBox mWarnings;

CheckBox mInfoWarnings;

CheckBox mVerbose;

CheckBox mVtls;

CheckBox mVgc;

CheckBox mUseDeprecated;

CheckBox mErrDeprecated;

CheckBox mIgnorePragmas;

CheckBox mCheckProperty;

}

class DmdDocPropertyPage : ProjectPropertyPage

{

override string GetCategoryName() { return "Compiler"; }

override string GetPageName() { return "Documentation"; }

override void CreateControls()

{

AddControl("", mGenDoc = new CheckBox(mCanvas, "Generate documentation"));

AddControl("Documentation file", mDocFile = new Text(mCanvas));

AddControl("Documentation dir", mDocDir = new Text(mCanvas));

AddControl("CanDyDOC module", mModulesDDoc = new Text(mCanvas));

AddControl("", mGenHdr = new CheckBox(mCanvas, "Generate interface headers"));

AddControl("Header file", mHdrFile = new Text(mCanvas));

AddControl("Header directory", mHdrDir = new Text(mCanvas));

AddControl("", mGenJSON = new CheckBox(mCanvas, "Generate JSON file"));

AddControl("JSON file", mJSONFile = new Text(mCanvas));

}

override void UpdateDirty(bool bDirty)

{

super.UpdateDirty(bDirty);

EnableControls();

}

void EnableControls()

{

mDocDir.setEnabled(mGenDoc.isChecked());

mDocFile.setEnabled(mGenDoc.isChecked());

mModulesDDoc.setEnabled(mGenDoc.isChecked());

mHdrDir.setEnabled(mGenHdr.isChecked());

mHdrFile.setEnabled(mGenHdr.isChecked());

mJSONFile.setEnabled(mGenJSON.isChecked());

}

override void SetControls(ProjectOptions options)

{

mGenDoc.setChecked(options.doDocComments);

mDocDir.setText(options.docdir);

mDocFile.setText(options.docname);

mModulesDDoc.setText(options.modules\_ddoc);

mGenHdr.setChecked(options.doHdrGeneration);

mHdrDir.setText(options.hdrdir);

mHdrFile.setText(options.hdrname);

mGenJSON.setChecked(options.doXGeneration);

mJSONFile.setText(options.xfilename);

EnableControls();

}

override int DoApply(ProjectOptions options, ProjectOptions refoptions)

{

int changes = 0;

changes += changeOption(mGenDoc.isChecked(), options.doDocComments, refoptions.doDocComments);

changes += changeOption(mDocDir.getText(), options.docdir, refoptions.docdir);

changes += changeOption(mDocFile.getText(), options.docname, refoptions.docname);

changes += changeOption(mModulesDDoc.getText(), options.modules\_ddoc, refoptions.modules\_ddoc);

changes += changeOption(mGenHdr.isChecked(), options.doHdrGeneration, refoptions.doHdrGeneration);

changes += changeOption(mHdrDir.getText(), options.hdrdir, refoptions.hdrdir);

changes += changeOption(mHdrFile.getText(), options.hdrname, refoptions.hdrname);

changes += changeOption(mGenJSON.isChecked(), options.doXGeneration, refoptions.doXGeneration);

changes += changeOption(mJSONFile.getText(), options.xfilename, refoptions.xfilename);

return changes;

}

CheckBox mGenDoc;

Text mDocDir;

Text mDocFile;

Text mModulesDDoc;

CheckBox mGenHdr;

Text mHdrDir;

Text mHdrFile;

CheckBox mGenJSON;

Text mJSONFile;

}

class DmdOutputPropertyPage : ProjectPropertyPage

{

override string GetCategoryName() { return "Compiler"; }

override string GetPageName() { return "Output"; }

override void CreateControls()

{

mUnindentCheckBox = kLabelWidth;

AddControl("", mMultiObj = new CheckBox(mCanvas, "Multiple Object Files"));

AddControl("", mPreservePaths = new CheckBox(mCanvas, "Keep Path From Source File"));

AddControl("", mMsCoff32 = new CheckBox(mCanvas, "Use MS-COFF object file format for Win32 (DMD 2.067+)"));

}

override void SetControls(ProjectOptions options)

{

mMultiObj.setChecked(options.multiobj);

mPreservePaths.setChecked(options.preservePaths);

mMsCoff32.setChecked(options.mscoff);

}

override int DoApply(ProjectOptions options, ProjectOptions refoptions)

{

int changes = 0;

changes += changeOption(mMultiObj.isChecked(), options.multiobj, refoptions.multiobj);

changes += changeOption(mPreservePaths.isChecked(), options.preservePaths, refoptions.preservePaths);

changes += changeOption(mMsCoff32.isChecked(), options.mscoff, refoptions.mscoff);

return changes;

}

CheckBox mMultiObj;

CheckBox mPreservePaths;

CheckBox mMsCoff32;

}

class DmdLinkerPropertyPage : ProjectPropertyPage

{

override string GetCategoryName() { return "Linker"; }

override string GetPageName() { return "General"; }

this()

{

kNeededLines = 11;

}

override void UpdateDirty(bool bDirty)

{

super.UpdateDirty(bDirty);

EnableControls();

}

enum ID\_OBJECTFILES = 1050;

enum ID\_LIBRARYFILES = 1051;

enum ID\_LIBRARYPATHS = 1052;

enum ID\_DEFFILE = 1053;

enum ID\_RESFILE = 1054;

extern(D) override void OnCommand(Widget w, int cmd)

{

switch(cmd)

{

case ID\_OBJECTFILES:

addBrowsePath(mObjFiles, false, GetProjectDir(), " ", "Add object file", "Object files\0\*.obj\0All Files\0\*.\*\0");

break;

case ID\_LIBRARYFILES:

addBrowsePath(mLibFiles, false, GetProjectDir(), " ", "Add library file", "Library files\0\*.lib\0All Files\0\*.\*\0");

break;

case ID\_LIBRARYPATHS:

addBrowsePath(mLibPaths, true, GetProjectDir(), " ", "Add library path");

break;

case ID\_DEFFILE:

if(auto file = browseFile(mCanvas.hwnd, "Select definition file", "Definition files\0\*.def\0All Files\0\*.\*\0", GetProjectDir()))

mDefFile.setText(makeRelative(file, GetProjectDir()));

break;

case ID\_RESFILE:

if(auto file = browseFile(mCanvas.hwnd, "Select resource file", "Resource files\0\*.res\0All Files\0\*.\*\0", GetProjectDir()))

mResFile.setText(makeRelative(file, GetProjectDir()));

break;

default:

break;

}

super.OnCommand(w, cmd);

}

override void CreateControls()

{

AddControl("Output File", mExeFile = new Text(mCanvas));

auto btn = new Button(mCanvas, "+", ID\_OBJECTFILES);

AddControl("Object Files", mObjFiles = new Text(mCanvas), btn);

btn = new Button(mCanvas, "+", ID\_LIBRARYFILES);

AddControl("Library Files", mLibFiles = new Text(mCanvas), btn);

btn = new Button(mCanvas, "+", ID\_LIBRARYPATHS);

AddControl("Library Search Path", mLibPaths = new Text(mCanvas), btn);

//AddControl("Library search paths only work if you have modified sc.ini to include DMD\_LIB!", null);

btn = new Button(mCanvas, "...", ID\_DEFFILE);

AddControl("Definition File", mDefFile = new Text(mCanvas), btn);

btn = new Button(mCanvas, "...", ID\_RESFILE);

AddControl("Resource File", mResFile = new Text(mCanvas), btn);

AddControl("Generate Map File", mGenMap = new ComboBox(mCanvas,

[ "Minimum", "Symbols By Address", "Standard", "Full", "With cross references" ], false));

AddControl("", mImplib = new CheckBox(mCanvas, "Create import library"));

AddControl("", mPrivatePhobos = new CheckBox(mCanvas, "Build and use local version of phobos with same compiler options"));

AddControl("", mUseStdLibPath = new CheckBox(mCanvas, "Use global and standard library search paths"));

AddControl("C Runtime", mCRuntime = new ComboBox(mCanvas, [ "None", "Static Release (LIBCMT)", "Static Debug (LIBCMTD)", "Dynamic Release (MSCVRT)", "Dynamic Debug (MSCVRTD)" ], false));

}

void EnableControls()

{

if(ProjectOptions options = GetProjectOptions())

mCRuntime.setEnabled(options.isX86\_64 || options.mscoff);

}

override void SetControls(ProjectOptions options)

{

mExeFile.setText(options.exefile);

mObjFiles.setText(options.objfiles);

mLibFiles.setText(options.libfiles);

mLibPaths.setText(options.libpaths);

mDefFile.setText(options.deffile);

mResFile.setText(options.resfile);

mGenMap.setSelection(options.mapverbosity);

mImplib.setChecked(options.createImplib);

mUseStdLibPath.setChecked(options.useStdLibPath);

mPrivatePhobos.setChecked(options.privatePhobos);

mCRuntime.setSelection(options.cRuntime);

EnableControls();

}

override int DoApply(ProjectOptions options, ProjectOptions refoptions)

{

int changes = 0;

changes += changeOption(mExeFile.getText(), options.exefile, refoptions.exefile);

changes += changeOption(mObjFiles.getText(), options.objfiles, refoptions.objfiles);

changes += changeOption(mLibFiles.getText(), options.libfiles, refoptions.libfiles);

changes += changeOption(mLibPaths.getText(), options.libpaths, refoptions.libpaths);

changes += changeOption(mDefFile.getText(), options.deffile, refoptions.deffile);

changes += changeOption(mResFile.getText(), options.resfile, refoptions.resfile);

changes += changeOption(cast(uint) mGenMap.getSelection(), options.mapverbosity, refoptions.mapverbosity);

changes += changeOption(mImplib.isChecked(), options.createImplib, refoptions.createImplib);

changes += changeOption(mUseStdLibPath.isChecked(), options.useStdLibPath, refoptions.useStdLibPath);

changes += changeOption(mPrivatePhobos.isChecked(), options.privatePhobos, refoptions.privatePhobos);

changes += changeOption(cast(uint) mCRuntime.getSelection(), options.cRuntime, refoptions.cRuntime);

return changes;

}

Text mExeFile;

Text mObjFiles;

Text mLibFiles;

Text mLibPaths;

Text mDefFile;

Text mResFile;

ComboBox mGenMap;

CheckBox mImplib;

CheckBox mUseStdLibPath;

CheckBox mPrivatePhobos;

ComboBox mCRuntime;

}

class DmdEventsPropertyPage : ProjectPropertyPage

{

override string GetCategoryName() { return ""; }

override string GetPageName() { return "Build Events"; }

override void CreateControls()

{

mLinesPerMultiLine = 5;

AddControl("Pre-Build Command", mPreCmd = new MultiLineText(mCanvas), 500);

AddControl("Post-Build Command", mPostCmd = new MultiLineText(mCanvas), 500);

Label lab = new Label(mCanvas, "Use \"if errorlevel 1 goto reportError\" to cancel on error");

lab.setRect(0, mLineY, getWidgetWidth(mCanvas, kPageWidth), kLineHeight);

addResizableWidget(lab, kAttachBottom);

}

override void SetControls(ProjectOptions options)

{

mPreCmd.setText(options.preBuildCommand);

mPostCmd.setText(options.postBuildCommand);

}

override int DoApply(ProjectOptions options, ProjectOptions refoptions)

{

int changes = 0;

changes += changeOption(mPreCmd.getText(), options.preBuildCommand, refoptions.preBuildCommand);

changes += changeOption(mPostCmd.getText(), options.postBuildCommand, refoptions.postBuildCommand);

return changes;

}

MultiLineText mPreCmd;

MultiLineText mPostCmd;

}

class DmdCmdLinePropertyPage : ProjectPropertyPage

{

override string GetCategoryName() { return ""; }

override string GetPageName() { return "Command line"; }

override void CreateControls()

{

mLinesPerMultiLine = 5;

AddControl("Command line", mCmdLine = new MultiLineText(mCanvas, "", 0, true), 500);

AddControl("Additional options", mAddOpt = new MultiLineText(mCanvas), 500);

}

override void OnConfigModified()

{

if(ProjectOptions options = GetProjectOptions())

if(mCmdLine && mCmdLine.hwnd)

mCmdLine.setText(options.buildCommandLine(true, true, true));

}

override void SetControls(ProjectOptions options)

{

mCmdLine.setText(options.buildCommandLine(true, true, true));

mAddOpt.setText(options.additionalOptions);

}

override int DoApply(ProjectOptions options, ProjectOptions refoptions)

{

int changes = 0;

changes += changeOption(mAddOpt.getText(), options.additionalOptions, refoptions.additionalOptions);

return changes;

}

MultiLineText mCmdLine;

MultiLineText mAddOpt;

}

class ConfigNodePropertyPage : ProjectPropertyPage

{

abstract void SetControls(CFileNode node);

abstract int DoApply(CFileNode node, CFileNode refnode, Config cfg);

override void SetControls(ProjectOptions options)

{

mNodes = GetSelectedNodes();

if(auto node = GetNode())

SetControls(node);

}

override int DoApply(ProjectOptions options, ProjectOptions refoptions)

{

return 0;

}

CHierNode[] GetSelectedNodes()

{

if(auto cfg = GetConfig()) // any config works

{

auto prj = cfg.GetProject();

CHierNode[] nodes;

prj.GetSelectedNodes(nodes);

return nodes;

}

return null;

}

CFileNode GetNode()

{

for(size\_t i = 0; i < mNodes.length; i++)

if(auto node = cast(CFileNode)mNodes[i])

return node;

return null;

}

override int IsPageDirty()

{

mixin(LogCallMix);

if(mWindow)

if(CFileNode node = GetNode())

{

Config cfg = GetConfig();

scope CFileNode n = newCom!CFileNode("");

return DoApply(n, node, cfg) > 0 ? S\_OK : S\_FALSE;

}

return S\_FALSE;

}

override int Apply()

{

mixin(LogCallMix);

if(CFileNode rnode = GetNode())

{

auto refnode = rnode.cloneDeep();

for(int i = 0; i < mObjects.length; i++)

{

auto config = ComPtr!(Config)(mObjects[i]);

if(config)

{

for(size\_t n = 0; n < mNodes.length; n++)

if(auto node = cast(CFileNode)mNodes[n])

{

DoApply(node, refnode, config);

if(CProjectNode pn = cast(CProjectNode) node.GetRootNode())

pn.SetProjectFileDirty(true);

}

}

return S\_OK;

}

}

return returnError(E\_FAIL);

}

CHierNode[] mNodes;

}

class FilePropertyPage : ConfigNodePropertyPage

{

override string GetCategoryName() { return ""; }

override string GetPageName() { return "File"; }

override void CreateControls()

{

mLinesPerMultiLine = 3;

AddControl("", mPerConfig = new CheckBox(mCanvas, "per Configuration Options (apply and reopen dialog to update)"));

AddControl("Build Tool", mTool = new ComboBox(mCanvas, [ "Auto", "DMD", kToolCpp, kToolResourceCompiler, "Custom", "None" ], false));

AddControl("Additional Options", mAddOpt = new Text(mCanvas));

AddControl("Build Command", mCustomCmd = new MultiLineText(mCanvas), 1000);

AddControl("Other Dependencies", mDependencies = new Text(mCanvas));

AddControl("Output File", mOutFile = new Text(mCanvas));

AddControl("", mLinkOut = new CheckBox(mCanvas, "Add output to link"));

AddControl("", mUptodateWithSameTime = new CheckBox(mCanvas, "Assume output up to date with same time as input"));

}

override void UpdateDirty(bool bDirty)

{

super.UpdateDirty(bDirty);

enableControls(mTool.getText());

}

void enableControls(string tool)

{

bool perConfigChanged = mInitPerConfig != mPerConfig.isChecked();

bool isCustom = (tool == "Custom");

bool isRc = (tool == kToolResourceCompiler);

bool isCpp = (tool == kToolCpp);

mTool.setEnabled(!perConfigChanged);

mCustomCmd.setEnabled(!perConfigChanged && isCustom);

mAddOpt.setEnabled(!perConfigChanged && (isRc || isCpp));

mDependencies.setEnabled(!perConfigChanged && (isCustom || isRc));

mOutFile.setEnabled(!perConfigChanged && isCustom);

mLinkOut.setEnabled(!perConfigChanged && isCustom);

mUptodateWithSameTime.setEnabled(!perConfigChanged && isCustom);

}

string GetCfgName()

{

return GetConfig().getCfgName();

}

override void SetControls(CFileNode node)

{

string cfgname = GetCfgName();

string tool = node.GetTool(cfgname);

if(tool.length == 0)

mTool.setSelection(0);

else

mTool.setSelection(mTool.findString(tool));

mInitPerConfig = node.GetPerConfigOptions();

mPerConfig.setChecked(mInitPerConfig);

mCustomCmd.setText(node.GetCustomCmd(cfgname));

mAddOpt.setText(node.GetAdditionalOptions(cfgname));

mDependencies.setText(node.GetDependencies(cfgname));

mOutFile.setText(node.GetOutFile(cfgname));

mLinkOut.setChecked(node.GetLinkOutput(cfgname));

mUptodateWithSameTime.setChecked(node.GetUptodateWithSameTime(cfgname));

enableControls(tool);

}

override int DoApply(CFileNode node, CFileNode refnode, Config cfg)

{

string cfgname = GetCfgName();

int changes = 0;

string tool = mTool.getText();

if(tool == "Auto")

tool = "";

changes += changeOptionDg!bool(mPerConfig.isChecked(), &node.SetPerConfigOptions, refnode.GetPerConfigOptions());

changes += changeOptionDg!string(tool, (s) => node.SetTool(cfgname, s), refnode.GetTool(cfgname));

changes += changeOptionDg!string(mCustomCmd.getText(), (s) => node.SetCustomCmd(cfgname, s), refnode.GetCustomCmd(cfgname));

changes += changeOptionDg!string(mAddOpt.getText(), (s) => node.SetAdditionalOptions(cfgname, s), refnode.GetAdditionalOptions(cfgname));

changes += changeOptionDg!string(mDependencies.getText(), (s) => node.SetDependencies(cfgname, s), refnode.GetDependencies(cfgname));

changes += changeOptionDg!string(mOutFile.getText(), (s) => node.SetOutFile(cfgname, s), refnode.GetOutFile(cfgname));

changes += changeOptionDg!bool(mLinkOut.isChecked(), (b) => node.SetLinkOutput(cfgname, b), refnode.GetLinkOutput(cfgname));

changes += changeOptionDg!bool(mUptodateWithSameTime.isChecked(),

(b) => node.SetUptodateWithSameTime(cfgname, b), refnode.GetUptodateWithSameTime(cfgname));

enableControls(tool);

return changes;

}

bool mInitPerConfig;

CheckBox mPerConfig;

ComboBox mTool;

MultiLineText mCustomCmd;

Text mAddOpt;

Text mDependencies;

Text mOutFile;

CheckBox mLinkOut;

CheckBox mUptodateWithSameTime;

}

///////////////////////////////////////////////////////////////////////////////

class DirPropertyPage : GlobalPropertyPage

{

enum ID\_BROWSEINSTALLDIR = 1000;

enum ID\_IMPORTDIR = 1001;

enum ID\_EXEPATH32 = 1002;

enum ID\_EXEPATH64 = 1003;

enum ID\_EXEPATH32COFF = 1004;

enum ID\_LIBPATH32 = 1005;

enum ID\_LIBPATH64 = 1006;

enum ID\_LIBPATH32COFF = 1007;

enum ID\_LINKER64 = 1008;

enum ID\_LINKER32COFF = 1009;

this(GlobalOptions options)

{

super(options);

kNeededLines = 13;

}

void addBrowseDir(MultiLineText ctrl, string title)

{

addBrowsePath(ctrl, true, null, "\n", title);

}

extern(D) override void OnCommand(Widget w, int cmd)

{

switch(cmd)

{

case ID\_BROWSEINSTALLDIR:

if(auto dir = browseDirectory(mCanvas.hwnd, "Select installation directory"))

mDmdPath.setText(dir);

break;

case ID\_IMPORTDIR:

addBrowseDir(mImpPath, "Add import directory");

break;

case ID\_EXEPATH32:

addBrowseDir(mExePath, "Add executable directory");

break;

case ID\_EXEPATH64:

addBrowseDir(mExePath64, "Add executable directory");

break;

case ID\_EXEPATH32COFF:

addBrowseDir(mExePath32coff, "Add executable directory");

break;

case ID\_LIBPATH32:

addBrowseDir(mLibPath, "Add library directory");

break;

case ID\_LIBPATH64:

addBrowseDir(mLibPath64, "Add library directory");

break;

case ID\_LIBPATH32COFF:

addBrowseDir(mLibPath32coff, "Add library directory");

break;

case ID\_LINKER64:

if(auto file = browseFile(mCanvas.hwnd, "Select linker executable", "Executables\0\*.exe\0All Files\0\*.\*\0"))

mLinkerExecutable64.setText(file);

break;

case ID\_LINKER32COFF:

if(auto file = browseFile(mCanvas.hwnd, "Select linker executable", "Executables\0\*.exe\0All Files\0\*.\*\0"))

mLinkerExecutable32coff.setText(file);

break;

default:

break;

}

super.OnCommand(w, cmd);

}

void dirCreateControls(string name, string overrideIni)

{

auto btn = new Button(mCanvas, "...", ID\_BROWSEINSTALLDIR);

AddControl(name ~ " install path", mDmdPath = new Text(mCanvas), btn);

mLinesPerMultiLine = 2;

btn = new Button(mCanvas, "+", ID\_IMPORTDIR);

AddControl("Import paths", mImpPath = new MultiLineText(mCanvas), btn, 300);

mLinesPerMultiLine = 10;

string[] archs = ["Win32", "x64"];

if(overrideIni.length)

archs ~= "Win32-COFF";

AddControl("", mTabArch = new TabControl(mCanvas, archs), 700);

auto page32 = mTabArch.pages[0];

if(auto w = cast(Window)page32)

w.commandDelegate = mCanvas.commandDelegate;

mLineY = 0;

mAttachY = 0;

mLinesPerMultiLine = 3;

btn = new Button(page32, "+", ID\_EXEPATH32);

AddControl("Executable paths", mExePath = new MultiLineText(page32), btn, 500);

mLinesPerMultiLine = 2;

btn = new Button(page32, "+", ID\_LIBPATH32);

AddControl("Library paths", mLibPath = new MultiLineText(page32), btn, 500);

AddControl("Disassemble Command", mDisasmCommand = new Text(page32));

auto page64 = mTabArch.pages[1];

if(auto w = cast(Window)page64)

w.commandDelegate = mCanvas.commandDelegate;

mLineY = 0;

mAttachY = 0;

mLinesPerMultiLine = 3;

btn = new Button(page64, "+", ID\_EXEPATH64);

AddControl("Executable paths", mExePath64 = new MultiLineText(page64), btn, 500);

mLinesPerMultiLine = 2;

btn = new Button(page64, "+", ID\_LIBPATH64);

AddControl("Library paths", mLibPath64 = new MultiLineText(page64), btn, 500);

AddControl("Disassemble Command", mDisasmCommand64 = new Text(page64));

if(overrideIni.length)

{

AddControl("", mOverrideIni64 = new CheckBox(page64, overrideIni));

btn = new Button(page64, "...", ID\_LINKER64);

AddControl("Linker", mLinkerExecutable64 = new Text(page64), btn);

AddControl("Additional options", mLinkerOptions64 = new Text(page64));

auto page32coff = mTabArch.pages[2];

if(auto w = cast(Window)page32coff)

w.commandDelegate = mCanvas.commandDelegate;

mLineY = 0;

mAttachY = 0;

mLinesPerMultiLine = 3;

btn = new Button(page32coff, "+", ID\_EXEPATH32COFF);

AddControl("Executable paths", mExePath32coff = new MultiLineText(page32coff), btn, 500);

mLinesPerMultiLine = 2;

btn = new Button(page32coff, "+", ID\_LIBPATH32COFF);

AddControl("Library paths", mLibPath32coff = new MultiLineText(page32coff), btn, 500);

AddControl("Disassemble Command", mDisasmCommand32coff = new Text(page32coff));

AddControl("", mOverrideIni32coff = new CheckBox(page32coff, overrideIni));

btn = new Button(page32coff, "...", ID\_LINKER32COFF);

AddControl("Linker", mLinkerExecutable32coff = new Text(page32coff), btn);

AddControl("Additional options", mLinkerOptions32coff = new Text(page32coff));

}

}

override void UpdateDirty(bool bDirty)

{

super.UpdateDirty(bDirty);

enableControls();

}

void enableControls()

{

if(mOverrideIni64)

{

mLinkerExecutable64.setEnabled(mOverrideIni64.isChecked());

mLinkerOptions64.setEnabled(mOverrideIni64.isChecked());

}

if(mOverrideIni32coff)

{

mLinkerExecutable32coff.setEnabled(mOverrideIni32coff.isChecked());

mLinkerOptions32coff.setEnabled(mOverrideIni32coff.isChecked());

}

}

abstract CompilerDirectories\* getCompilerOptions(GlobalOptions opts);

override void SetControls(GlobalOptions opts)

{

CompilerDirectories\* opt = getCompilerOptions(opts);

mDmdPath.setText(opt.InstallDir);

mExePath.setText(opt.ExeSearchPath);

mImpPath.setText(opt.ImpSearchPath);

mLibPath.setText(opt.LibSearchPath);

mDisasmCommand.setText(opt.DisasmCommand);

mExePath64.setText(opt.ExeSearchPath64);

mLibPath64.setText(opt.LibSearchPath64);

mDisasmCommand64.setText(opt.DisasmCommand64);

if(mOverrideIni64)

{

mOverrideIni64.setChecked(opt.overrideIni64);

mLinkerExecutable64.setText(opt.overrideLinker64);

mLinkerOptions64.setText(opt.overrideOptions64);

}

if(mOverrideIni32coff)

{

mExePath32coff.setText(opt.ExeSearchPath32coff);

mLibPath32coff.setText(opt.LibSearchPath32coff);

mOverrideIni32coff.setChecked(opt.overrideIni32coff);

mLinkerExecutable32coff.setText(opt.overrideLinker32coff);

mLinkerOptions32coff.setText(opt.overrideOptions32coff);

mDisasmCommand32coff.setText(opt.DisasmCommand32coff);

}

enableControls();

}

override int DoApply(GlobalOptions opts, GlobalOptions refopts)

{

CompilerDirectories\* opt = getCompilerOptions(opts);

CompilerDirectories\* refopt = getCompilerOptions(refopts);

int changes = 0;

changes += changeOption(mDmdPath.getText(), opt.InstallDir, refopt.InstallDir);

changes += changeOption(mExePath.getText(), opt.ExeSearchPath, refopt.ExeSearchPath);

changes += changeOption(mImpPath.getText(), opt.ImpSearchPath, refopt.ImpSearchPath);

changes += changeOption(mLibPath.getText(), opt.LibSearchPath, refopt.LibSearchPath);

changes += changeOption(mDisasmCommand.getText(), opt.DisasmCommand, refopt.DisasmCommand);

changes += changeOption(mExePath64.getText(), opt.ExeSearchPath64, refopt.ExeSearchPath64);

changes += changeOption(mLibPath64.getText(), opt.LibSearchPath64, refopt.LibSearchPath64);

changes += changeOption(mDisasmCommand64.getText(), opt.DisasmCommand64, refopt.DisasmCommand64);

if(mOverrideIni64)

{

changes += changeOption(mOverrideIni64.isChecked(), opt.overrideIni64, refopt.overrideIni64);

changes += changeOption(mLinkerExecutable64.getText(), opt.overrideLinker64, refopt.overrideLinker64);

changes += changeOption(mLinkerOptions64.getText(), opt.overrideOptions64, refopt.overrideOptions64);

}

if(mOverrideIni32coff)

{

changes += changeOption(mExePath32coff.getText(), opt.ExeSearchPath32coff, refopt.ExeSearchPath32coff);

changes += changeOption(mLibPath32coff.getText(), opt.LibSearchPath32coff, refopt.LibSearchPath32coff);

changes += changeOption(mOverrideIni32coff.isChecked(), opt.overrideIni32coff, refopt.overrideIni32coff);

changes += changeOption(mLinkerExecutable32coff.getText(), opt.overrideLinker32coff, refopt.overrideLinker32coff);

changes += changeOption(mLinkerOptions32coff.getText(), opt.overrideOptions32coff, refopt.overrideOptions32coff);

changes += changeOption(mDisasmCommand32coff.getText(), opt.DisasmCommand32coff, refopt.DisasmCommand32coff);

}

return changes;

}

TabControl mTabArch;

Text mDmdPath;

MultiLineText mExePath;

MultiLineText mImpPath;

MultiLineText mLibPath;

Text mDisasmCommand;

MultiLineText mExePath64;

MultiLineText mLibPath64;

CheckBox mOverrideIni64;

Text mLinkerExecutable64;

Text mLinkerOptions64;

Text mDisasmCommand64;

MultiLineText mExePath32coff;

MultiLineText mLibPath32coff;

CheckBox mOverrideIni32coff;

Text mLinkerExecutable32coff;

Text mLinkerOptions32coff;

Text mDisasmCommand32coff;

}

///////////////////////////////////////////////////////////////////////////////

class DmdDirPropertyPage : DirPropertyPage

{

override string GetCategoryName() { return "D Options"; }

override string GetPageName() { return "DMD Directories"; }

this(GlobalOptions options)

{

super(options);

}

override void CreateControls()

{

dirCreateControls("DMD", "override linker settings from dmd configuration in sc.ini.");

}

override CompilerDirectories\* getCompilerOptions(GlobalOptions opts)

{

return &opts.DMD;

}

}

///////////////////////////////////////////////////////////////////////////////

class GdcDirPropertyPage : DirPropertyPage

{

override string GetCategoryName() { return "D Options"; }

override string GetPageName() { return "GDC Directories"; }

this(GlobalOptions options)

{

super(options);

}

override void CreateControls()

{

dirCreateControls("GDC", "");

}

override CompilerDirectories\* getCompilerOptions(GlobalOptions opts)

{

return &opts.GDC;

}

}

///////////////////////////////////////////////////////////////////////////////

class LdcDirPropertyPage : DirPropertyPage

{

override string GetCategoryName() { return "D Options"; }

override string GetPageName() { return "LDC Directories"; }

this(GlobalOptions options)

{

super(options);

}

override void CreateControls()

{

dirCreateControls("LDC", "");

}

override CompilerDirectories\* getCompilerOptions(GlobalOptions opts)

{

return &opts.LDC;

}

}

///////////////////////////////////////////////////////////////////////////////

class ToolsProperty2Page : GlobalPropertyPage

{

override string GetCategoryName() { return "Projects"; }

override string GetPageName() { return "D Options"; }

this(GlobalOptions options)

{

super(options);

kNeededLines = 13;

}

override void CreateControls()

{

AddControl("", mSortProjects = new CheckBox(mCanvas, "Sort project items"));

AddControl("", mShowUptodate = new CheckBox(mCanvas, "Show why a target is rebuilt"));

AddControl("", mTimeBuilds = new CheckBox(mCanvas, "Show build time"));

AddControl("", mStopSlnBuild = new CheckBox(mCanvas, "Stop solution build on error"));

AddHorizontalLine();

AddControl("", mDemangleError = new CheckBox(mCanvas, "Demangle names in link errors/disassembly"));

AddControl("", mOptlinkDeps = new CheckBox(mCanvas, "Monitor linker dependencies"));

AddHorizontalLine();

//AddControl("Remove project item", mDeleteFiles =

//                 new ComboBox(mCanvas, [ "Do not delete file on disk", "Ask", "Delete file on disk" ]));

mLinesPerMultiLine = 2;

AddControl("JSON paths", mJSNPath = new MultiLineText(mCanvas));

AddControl("Resource includes", mIncPath = new Text(mCanvas));

AddHorizontalLine();

AddControl("Compile+Run options", mCompileAndRunOpts = new Text(mCanvas));

AddControl("Compile+Debug options", mCompileAndDbgOpts = new Text(mCanvas));

AddControl(" Debugger", mCompileAndDbgEngine = new ComboBox(mCanvas, [ "Visual Studio", "Mago", "Visual Studio (x86 Mixed Mode)" ], false));

}

override void SetControls(GlobalOptions opts)

{

mTimeBuilds.setChecked(opts.timeBuilds);

mSortProjects.setChecked(opts.sortProjects);

mShowUptodate.setChecked(opts.showUptodateFailure);

mStopSlnBuild.setChecked(opts.stopSolutionBuild);

mDemangleError.setChecked(opts.demangleError);

mOptlinkDeps.setChecked(opts.optlinkDeps);

//mDeleteFiles.setSelection(opts.deleteFiles + 1);

mIncPath.setText(opts.IncSearchPath);

mJSNPath.setText(opts.JSNSearchPath);

mCompileAndRunOpts.setText(opts.compileAndRunOpts);

mCompileAndDbgOpts.setText(opts.compileAndDbgOpts);

mCompileAndDbgEngine.setSelection(opts.compileAndDbgEngine);

}

override int DoApply(GlobalOptions opts, GlobalOptions refopts)

{

int changes = 0;

changes += changeOption(mTimeBuilds.isChecked(), opts.timeBuilds, refopts.timeBuilds);

changes += changeOption(mSortProjects.isChecked(), opts.sortProjects, refopts.sortProjects);

changes += changeOption(mShowUptodate.isChecked(), opts.showUptodateFailure, refopts.showUptodateFailure);

changes += changeOption(mStopSlnBuild.isChecked(), opts.stopSolutionBuild, refopts.stopSolutionBuild);

changes += changeOption(mDemangleError.isChecked(), opts.demangleError, refopts.demangleError);

changes += changeOption(mOptlinkDeps.isChecked(), opts.optlinkDeps, refopts.optlinkDeps);

//changes += changeOption(cast(byte) (mDeleteFiles.getSelection() - 1), opts.deleteFiles, refopts.deleteFiles);

changes += changeOption(mIncPath.getText(), opts.IncSearchPath, refopts.IncSearchPath);

changes += changeOption(mJSNPath.getText(), opts.JSNSearchPath, refopts.JSNSearchPath);

changes += changeOption(mCompileAndRunOpts.getText(), opts.compileAndRunOpts, refopts.compileAndRunOpts);

changes += changeOption(mCompileAndDbgOpts.getText(), opts.compileAndDbgOpts, refopts.compileAndDbgOpts);

changes += changeOption(mCompileAndDbgEngine.getSelection(), opts.compileAndDbgEngine, refopts.compileAndDbgEngine);

return changes;

}

CheckBox mTimeBuilds;

CheckBox mSortProjects;

CheckBox mShowUptodate;

CheckBox mStopSlnBuild;

CheckBox mDemangleError;

CheckBox mOptlinkDeps;

//ComboBox mDeleteFiles;

Text mIncPath;

Text mCompileAndRunOpts;

Text mCompileAndDbgOpts;

ComboBox mCompileAndDbgEngine;

MultiLineText mJSNPath;

}

///////////////////////////////////////////////////////////////////////////////

class ColorizerPropertyPage : GlobalPropertyPage

{

override string GetCategoryName() { return "Language"; }

override string GetPageName() { return "Colorizer"; }

this(GlobalOptions options)

{

super(options);

kNeededLines = 11;

}

override void CreateControls()

{

AddControl("", mColorizeVersions = new CheckBox(mCanvas, "Colorize version and debug statements"));

AddControl("Colored types", mUserTypes = new MultiLineText(mCanvas), 1000);

AddHorizontalLine();

AddControl("", mColorizeCoverage = new CheckBox(mCanvas, "Colorize coverage from .LST file"));

AddControl("", mShowCoverageMargin = new CheckBox(mCanvas, "Show coverage margin"));

AddHorizontalLine();

AddControl("", mAutoOutlining = new CheckBox(mCanvas, "Add outlining regions when opening D files"));

AddControl("", mParseSource = new CheckBox(mCanvas, "Parse source for syntax errors"));

AddControl("", mPasteIndent = new CheckBox(mCanvas, "Reindent new lines after paste"));

}

override void SetControls(GlobalOptions opts)

{

mColorizeVersions.setChecked(opts.ColorizeVersions);

mColorizeCoverage.setChecked(opts.ColorizeCoverage);

mShowCoverageMargin.setChecked(opts.showCoverageMargin);

mAutoOutlining.setChecked(opts.autoOutlining);

mParseSource.setChecked(opts.parseSource);

mPasteIndent.setChecked(opts.pasteIndent);

mUserTypes.setText(opts.UserTypesSpec);

//mSemantics.setEnabled(false);

}

override int DoApply(GlobalOptions opts, GlobalOptions refopts)

{

int changes = 0;

changes += changeOption(mColorizeVersions.isChecked(), opts.ColorizeVersions, refopts.ColorizeVersions);

changes += changeOption(mColorizeCoverage.isChecked(), opts.ColorizeCoverage, refopts.ColorizeCoverage);

changes += changeOption(mShowCoverageMargin.isChecked(), opts.showCoverageMargin, refopts.showCoverageMargin);

changes += changeOption(mAutoOutlining.isChecked(), opts.autoOutlining, refopts.autoOutlining);

changes += changeOption(mParseSource.isChecked(), opts.parseSource, refopts.parseSource);

changes += changeOption(mPasteIndent.isChecked(), opts.pasteIndent, refopts.pasteIndent);

changes += changeOption(mUserTypes.getText(), opts.UserTypesSpec, refopts.UserTypesSpec);

return changes;

}

CheckBox mColorizeVersions;

CheckBox mColorizeCoverage;

CheckBox mShowCoverageMargin;

CheckBox mAutoOutlining;

CheckBox mParseSource;

CheckBox mPasteIndent;

MultiLineText mUserTypes;

}

///////////////////////////////////////////////////////////////////////////////

class IntellisensePropertyPage : GlobalPropertyPage

{

override string GetCategoryName() { return "Language"; }

override string GetPageName() { return "Intellisense"; }

this(GlobalOptions options)

{

super(options);

}

override void CreateControls()

{

AddControl("", mExpandSemantics = new CheckBox(mCanvas, "Expansions from semantic analysis"));

AddControl("", mExpandFromBuffer = new CheckBox(mCanvas, "Expansions from text buffer"));

AddControl("", mExpandFromJSON = new CheckBox(mCanvas, "Expansions from JSON browse information"));

AddControl("Show expansion when", mExpandTrigger = new ComboBox(mCanvas, [ "pressing Ctrl+Space", "writing '.'", "writing an identifier" ], false));

AddControl("", mShowTypeInTooltip = new CheckBox(mCanvas, "Show type of expressions in tool tip"));

AddControl("", mSemanticGotoDef = new CheckBox(mCanvas, "Use semantic analysis for \"Goto Definition\" (before trying JSON info)"));

version(DParserOption) AddControl("", mUseDParser = new CheckBox(mCanvas, "Use Alexander Bothe's D parsing engine for semantic analysis"));

AddControl("", mMixinAnalysis = new CheckBox(mCanvas, "Enable mixin analysis"));

AddControl("", mUFCSExpansions = new CheckBox(mCanvas, "Enable UFCS expansions"));

}

override void UpdateDirty(bool bDirty)

{

super.UpdateDirty(bDirty);

EnableControls();

}

void EnableControls()

{

version(DParserOption) bool useDParser = mUseDParser.isChecked();

else bool useDParser = true;

mMixinAnalysis.setEnabled(useDParser);

mUFCSExpansions.setEnabled(useDParser);

}

override void SetControls(GlobalOptions opts)

{

mExpandSemantics.setChecked(opts.expandFromSemantics);

mExpandFromBuffer.setChecked(opts.expandFromBuffer);

mExpandFromJSON.setChecked(opts.expandFromJSON);

mExpandTrigger.setSelection(opts.expandTrigger);

mShowTypeInTooltip.setChecked(opts.showTypeInTooltip);

mSemanticGotoDef.setChecked(opts.semanticGotoDef);

version(DParserOption) mUseDParser.setChecked(opts.useDParser);

mMixinAnalysis.setChecked(opts.mixinAnalysis);

mUFCSExpansions.setChecked(opts.UFCSExpansions);

//mExpandSemantics.setEnabled(false);

}

override int DoApply(GlobalOptions opts, GlobalOptions refopts)

{

int changes = 0;

changes += changeOption(mExpandSemantics.isChecked(), opts.expandFromSemantics, refopts.expandFromSemantics);

changes += changeOption(mExpandFromBuffer.isChecked(), opts.expandFromBuffer, refopts.expandFromBuffer);

changes += changeOption(mExpandFromJSON.isChecked(), opts.expandFromJSON, refopts.expandFromJSON);

changes += changeOption(cast(byte) mExpandTrigger.getSelection(), opts.expandTrigger, refopts.expandTrigger);

changes += changeOption(mShowTypeInTooltip.isChecked(), opts.showTypeInTooltip, refopts.showTypeInTooltip);

changes += changeOption(mSemanticGotoDef.isChecked(), opts.semanticGotoDef, refopts.semanticGotoDef);

version(DParserOption) changes += changeOption(mUseDParser.isChecked(), opts.useDParser, refopts.useDParser);

changes += changeOption(mMixinAnalysis.isChecked(), opts.mixinAnalysis, refopts.mixinAnalysis);

changes += changeOption(mUFCSExpansions.isChecked(), opts.UFCSExpansions, refopts.UFCSExpansions);

return changes;

}

CheckBox mExpandSemantics;

CheckBox mExpandFromBuffer;

CheckBox mExpandFromJSON;

ComboBox mExpandTrigger;

CheckBox mShowTypeInTooltip;

CheckBox mSemanticGotoDef;

version(DParserOption) CheckBox mUseDParser;

CheckBox mUFCSExpansions;

CheckBox mMixinAnalysis;

}

///////////////////////////////////////////////////////////////////////////////

// more guids in dpackage.d starting up to 980f

const GUID g\_GeneralPropertyPage = uuid("002a2de9-8bb6-484d-9810-7e4ad4084715");

const GUID g\_DmdGeneralPropertyPage = uuid("002a2de9-8bb6-484d-9811-7e4ad4084715");

const GUID g\_DmdDebugPropertyPage = uuid("002a2de9-8bb6-484d-9812-7e4ad4084715");

const GUID g\_DmdCodeGenPropertyPage = uuid("002a2de9-8bb6-484d-9813-7e4ad4084715");

const GUID g\_DmdMessagesPropertyPage = uuid("002a2de9-8bb6-484d-9814-7e4ad4084715");

const GUID g\_DmdOutputPropertyPage = uuid("002a2de9-8bb6-484d-9815-7e4ad4084715");

const GUID g\_DmdLinkerPropertyPage = uuid("002a2de9-8bb6-484d-9816-7e4ad4084715");

const GUID g\_DmdEventsPropertyPage = uuid("002a2de9-8bb6-484d-9817-7e4ad4084715");

const GUID g\_CommonPropertyPage = uuid("002a2de9-8bb6-484d-9818-7e4ad4084715");

const GUID g\_DebuggingPropertyPage = uuid("002a2de9-8bb6-484d-9819-7e4ad4084715");

const GUID g\_FilePropertyPage = uuid("002a2de9-8bb6-484d-981a-7e4ad4084715");

const GUID g\_DmdDocPropertyPage = uuid("002a2de9-8bb6-484d-981b-7e4ad4084715");

const GUID g\_DmdCmdLinePropertyPage = uuid("002a2de9-8bb6-484d-981c-7e4ad4084715");

// does not need to be registered, created explicitely by package

const GUID g\_DmdDirPropertyPage = uuid("002a2de9-8bb6-484d-9820-7e4ad4084715");

const GUID g\_GdcDirPropertyPage = uuid("002a2de9-8bb6-484d-9824-7e4ad4084715");

const GUID g\_LdcDirPropertyPage = uuid("002a2de9-8bb6-484d-9825-7e4ad4084715");

const GUID g\_ToolsProperty2Page = uuid("002a2de9-8bb6-484d-9822-7e4ad4084715");

// registered under Languages\\Language Services\\D\\EditorToolsOptions\\Colorizer, created explicitely by package

const GUID g\_ColorizerPropertyPage = uuid("002a2de9-8bb6-484d-9821-7e4ad4084715");

const GUID g\_IntellisensePropertyPage = uuid("002a2de9-8bb6-484d-9823-7e4ad4084715");

const GUID\*[] guids\_propertyPages =

[

&g\_GeneralPropertyPage,

&g\_DmdGeneralPropertyPage,

&g\_DmdDebugPropertyPage,

&g\_DmdCodeGenPropertyPage,

&g\_DmdMessagesPropertyPage,

&g\_DmdOutputPropertyPage,

&g\_DmdLinkerPropertyPage,

&g\_DmdEventsPropertyPage,

&g\_CommonPropertyPage,

&g\_DebuggingPropertyPage,

&g\_FilePropertyPage,

&g\_DmdDocPropertyPage,

&g\_DmdCmdLinePropertyPage,

];

class PropertyPageFactory : DComObject, IClassFactory

{

static PropertyPageFactory create(CLSID\* rclsid)

{

foreach(id; guids\_propertyPages)

if(\*id == \*rclsid)

return newCom!PropertyPageFactory(rclsid);

return null;

}

this(CLSID\* rclsid)

{

mClsid = \*rclsid;

}

override HRESULT QueryInterface(in IID\* riid, void\*\* pvObject)

{

if(queryInterface2!(IClassFactory) (this, IID\_IClassFactory, riid, pvObject))

return S\_OK;

return super.QueryInterface(riid, pvObject);

}

override HRESULT CreateInstance(IUnknown UnkOuter, in IID\* riid, void\*\* pvObject)

{

PropertyPage ppp;

assert(!UnkOuter);

if(mClsid == g\_GeneralPropertyPage)

ppp = newCom!GeneralPropertyPage();

else if(mClsid == g\_DebuggingPropertyPage)

ppp = newCom!DebuggingPropertyPage();

else if(mClsid == g\_DmdGeneralPropertyPage)

ppp = newCom!DmdGeneralPropertyPage();

else if(mClsid == g\_DmdDebugPropertyPage)

ppp = newCom!DmdDebugPropertyPage();

else if(mClsid == g\_DmdCodeGenPropertyPage)

ppp = newCom!DmdCodeGenPropertyPage();

else if(mClsid == g\_DmdMessagesPropertyPage)

ppp = newCom!DmdMessagesPropertyPage();

else if(mClsid == g\_DmdDocPropertyPage)

ppp = newCom!DmdDocPropertyPage();

else if(mClsid == g\_DmdOutputPropertyPage)

ppp = newCom!DmdOutputPropertyPage();

else if(mClsid == g\_DmdLinkerPropertyPage)

ppp = newCom!DmdLinkerPropertyPage();

else if(mClsid == g\_DmdEventsPropertyPage)

ppp = newCom!DmdEventsPropertyPage();

else if(mClsid == g\_DmdCmdLinePropertyPage)

ppp = newCom!DmdCmdLinePropertyPage();

else if(mClsid == g\_CommonPropertyPage)

ppp = newCom!CommonPropertyPage();

else if(mClsid == g\_FilePropertyPage)

ppp = newCom!FilePropertyPage();

else

return E\_INVALIDARG;

return ppp.QueryInterface(riid, pvObject);

}

override HRESULT LockServer(in BOOL fLock)

{

return S\_OK;

}

static int GetProjectPages(CAUUID \*pPages, bool addFile)

{

version(all) {

pPages.cElems = (addFile ? 12 : 11);

pPages.pElems = cast(GUID\*)CoTaskMemAlloc(pPages.cElems\*GUID.sizeof);

if (!pPages.pElems)

return E\_OUTOFMEMORY;

int idx = 0;

if(addFile)

pPages.pElems[idx++] = g\_FilePropertyPage;

pPages.pElems[idx++] = g\_GeneralPropertyPage;

pPages.pElems[idx++] = g\_DebuggingPropertyPage;

pPages.pElems[idx++] = g\_DmdGeneralPropertyPage;

pPages.pElems[idx++] = g\_DmdDebugPropertyPage;

pPages.pElems[idx++] = g\_DmdCodeGenPropertyPage;

pPages.pElems[idx++] = g\_DmdMessagesPropertyPage;

pPages.pElems[idx++] = g\_DmdDocPropertyPage;

pPages.pElems[idx++] = g\_DmdOutputPropertyPage;

pPages.pElems[idx++] = g\_DmdLinkerPropertyPage;

pPages.pElems[idx++] = g\_DmdCmdLinePropertyPage;

pPages.pElems[idx++] = g\_DmdEventsPropertyPage;

return S\_OK;

} else {

return returnError(E\_NOTIMPL);

}

}

static int GetCommonPages(CAUUID \*pPages)

{

pPages.cElems = 1;

pPages.pElems = cast(GUID\*)CoTaskMemAlloc(pPages.cElems\*GUID.sizeof);

if (!pPages.pElems)

return E\_OUTOFMEMORY;

pPages.pElems[0] = g\_CommonPropertyPage;

return S\_OK;

}

static int GetFilePages(CAUUID \*pPages)

{

pPages.cElems = 1;

pPages.pElems = cast(GUID\*)CoTaskMemAlloc(pPages.cElems\*GUID.sizeof);

if (!pPages.pElems)

return E\_OUTOFMEMORY;

pPages.pElems[0] = g\_FilePropertyPage;

return S\_OK;

}

private:

GUID mClsid;

}

// This file is part of Visual D

//

// Visual D integrates the D programming language into Visual Studio

// Copyright (c) 2010 by Rainer Schuetze, All Rights Reserved

//

// Distributed under the Boost Software License, Version 1.0.

// See accompanying file LICENSE\_1\_0.txt or copy at <http://www.boost.org/LICENSE_1_0.txt>

module visuald.register;

import visuald.windows;

import sdk.win32.winreg;

import std.string;

import std.conv;

import std.utf;

import std.path;

import std.file;

import std.datetime;

import std.array;

import stdext.string;

import stdext.registry;

import visuald.dpackage;

import visuald.dllmain;

import visuald.propertypage;

import visuald.config;

import visuald.comutil;

// Registers COM objects normally and registers VS Packages to the specified VS registry hive under HKCU

extern(Windows)

HRESULT VSDllRegisterServerUser(in wchar\* strRegRoot)

{

return VSDllRegisterServerInternal(strRegRoot, true);

}

// Unregisters COM objects normally and unregisters VS Packages from the specified VS registry hive under HKCU

extern(Windows)

HRESULT VSDllUnregisterServerUser(in wchar\* strRegRoot)

{

return VSDllUnregisterServerInternal(strRegRoot, true);

}

// Registers COM objects normally and registers VS Packages to the specified VS registry hive

extern(Windows)

HRESULT VSDllRegisterServer(in wchar\* strRegRoot)

{

return VSDllRegisterServerInternal(strRegRoot, false);

}

// Unregisters COM objects normally and unregisters VS Packages from the specified VS registry hive

extern(Windows)

HRESULT VSDllUnregisterServer(in wchar\* strRegRoot)

{

return VSDllUnregisterServerInternal(strRegRoot, false);

}

// Registers COM objects normally and registers VS Packages to the default VS registry hive

extern(Windows)

HRESULT DllRegisterServer()

{

return VSDllRegisterServer(null);

}

// Unregisters COM objects normally and unregisters VS Packages from the default VS registry hive

extern(Windows)

HRESULT DllUnregisterServer()

{

return VSDllUnregisterServer(null);

}

extern(Windows)

HRESULT WriteExtensionPackageDefinition(in wchar\* args)

{

wstring wargs = to\_wstring(args);

auto idx = indexOf(wargs, ' ');

if(idx < 1)

return E\_FAIL;

registryDump = "Windows Registry Editor Version 5.00\n"w;

registryRoot = (wargs[0 .. idx] ~ "\0"w)[0 .. idx];

string fname = to!string(wargs[idx + 1 .. $]);

try

{

HRESULT rc = VSDllRegisterServerInternal(registryRoot.ptr, false);

if(rc != S\_OK)

return rc;

string dir = dirName(fname);

if(!exists(dir))

mkdirRecurse(dir);

std.file.write(fname, (cast(wchar) 0xfeff) ~ registryDump); // add BOM

return S\_OK;

}

catch(Exception e)

{

MessageBox(null, toUTF16z(e.msg), args, MB\_OK);

}

return E\_FAIL;

}

///////////////////////////////////////////////////////////////////////

wstring registryDump;

wstring registryRoot;

class RegistryException : Exception

{

this(HRESULT hr)

{

super("Registry Error");

result = hr;

}

HRESULT result;

}

class RegKey

{

this(HKEY root, wstring keyname, bool write = true, bool chkDump = true, bool x64hive = false)

{

Create(root, keyname, write, chkDump, x64hive);

}

~this()

{

Close();

}

void Close()

{

if(key)

{

RegCloseKey(key);

key = null;

}

}

static wstring registryName(wstring name)

{

if(name.length == 0)

return "@"w;

return "\""w ~ escapeString(name) ~ "\""w;

}

void Create(HKEY root, wstring keyname, bool write = true, bool chkDump = true, bool x64hive = false)

{

HRESULT hr;

if(write && chkDump && registryRoot.length && keyname.startsWith(registryRoot))

{

if (keyname.startsWith(registryRoot))

registryDump ~= "\n[$RootKey$"w ~ keyname[registryRoot.length..$] ~ "]\n"w;

else

registryDump ~= "\n[\\"w ~ keyname ~ "]\n"w;

}

else if(write)

{

auto opt = REG\_OPTION\_NON\_VOLATILE | (x64hive ? KEY\_WOW64\_64KEY : 0);

hr = hrRegCreateKeyEx(root, keyname, 0, null, opt, KEY\_WRITE, null, &key, null);

if(FAILED(hr))

throw new RegistryException(hr);

}

else

hr = hrRegOpenKeyEx(root, keyname, (x64hive ? KEY\_WOW64\_64KEY : 0), KEY\_READ, &key);

}

void Set(wstring name, wstring value, bool escape = true)

{

if(!key && registryRoot.length)

{

if(escape)

value = escapeString(value);

registryDump ~= registryName(name) ~ "=\""w ~ value ~ "\"\n"w;

return;

}

if(!key)

throw new RegistryException(E\_FAIL);

HRESULT hr = RegCreateValue(key, name, value);

if(FAILED(hr))

throw new RegistryException(hr);

}

void Set(wstring name, uint value)

{

if(!key && registryRoot.length)

{

registryDump ~= registryName(name) ~ "=dword:"w;

registryDump ~= to!wstring(format("%08x", value)) ~ "\n";

return;

}

if(!key)

throw new RegistryException(E\_FAIL);

HRESULT hr = RegCreateDwordValue(key, name, value);

if(FAILED(hr))

throw new RegistryException(hr);

}

void Set(wstring name, long value)

{

if(!key && registryRoot.length)

{

registryDump ~= registryName(name) ~ "=qword:"w;

registryDump ~= to!wstring(to!string(value, 16) ~ "\n");

return;

}

if(!key)

throw new RegistryException(E\_FAIL);

HRESULT hr = RegCreateQwordValue(key, name, value);

if(FAILED(hr))

throw new RegistryException(hr);

}

void Set(wstring name, void[] data)

{

if(!key)

throw new RegistryException(E\_FAIL);

HRESULT hr = RegCreateBinaryValue(key, name, data);

if(FAILED(hr))

throw new RegistryException(hr);

}

bool Delete(wstring name)

{

if(!key && registryRoot.length)

return true; // ignore

if(!key)

return false;

wchar\* szName = \_toUTF16zw(name);

HRESULT hr = RegDeleteValue(key, szName);

return SUCCEEDED(hr);

}

wstring GetString(wstring name, wstring def = "")

{

if(!key)

return def;

wchar[260] buf;

DWORD cnt = 260 \* wchar.sizeof;

wchar\* szName = \_toUTF16zw(name);

DWORD type;

int hr = RegQueryValueExW(key, szName, null, &type, cast(ubyte\*) buf.ptr, &cnt);

if(hr == S\_OK && cnt > 0)

return to\_wstring(buf.ptr);

if(hr != ERROR\_MORE\_DATA || type != REG\_SZ)

return def;

scope wchar[] pbuf = new wchar[cnt/2 + 1];

RegQueryValueExW(key, szName, null, &type, cast(ubyte\*) pbuf.ptr, &cnt);

return to\_wstring(pbuf.ptr);

}

DWORD GetDWORD(wstring name, DWORD def = 0)

{

if(!key)

return def;

DWORD dw, type, cnt = dw.sizeof;

wchar\* szName = \_toUTF16zw(name);

int hr = RegQueryValueExW(key, szName, null, &type, cast(ubyte\*) &dw, &cnt);

if(hr != S\_OK || type != REG\_DWORD)

return def;

return dw;

}

void[] GetBinary(wstring name)

{

if(!key)

return null;

wchar\* szName = \_toUTF16zw(name);

DWORD type, cnt = 0;

int hr = RegQueryValueExW(key, szName, null, &type, cast(ubyte\*) &type, &cnt);

if(hr != ERROR\_MORE\_DATA || type != REG\_BINARY)

return null;

ubyte[] data = new ubyte[cnt];

hr = RegQueryValueExW(key, szName, null, &type, data.ptr, &cnt);

if(hr != S\_OK)

return null;

return data;

}

HKEY key;

}

///////////////////////////////////////////////////////////////////////

// convention: no trailing "\" for keys

static const wstring regPathConfigDefault = "Software\\Microsoft\\VisualStudio\\9.0"w;

static const wstring regPathFileExts = "[\\Languages\\File](file:///\\Languages\File) Extensions"w;

static const wstring regPathLServices = "[\\Languages\\Language](file:///\\Languages\Language) Services"w;

static const wstring regPathCodeExpansions = "[\\Languages\\CodeExpansions"w](file:///\\Languages\CodeExpansions%22w);

static const wstring regPathPrjTemplates = "[\\NewProjectTemplates\\TemplateDirs"w](file:///\\NewProjectTemplates\TemplateDirs%22w);

static const wstring regPathProjects = "\\Projects"w;

static const wstring regPathToolsOptions = "[\\ToolsOptionsPages\\Projects\\Visual](file:///\\ToolsOptionsPages\Projects\Visual) D Settings"w;

static const wstring regPathToolsDirsOld = "[\\ToolsOptionsPages\\Projects\\Visual](file:///\\ToolsOptionsPages\Projects\Visual) D Directories"w;

static const wstring regPathToolsDirsDmd = "[\\ToolsOptionsPages\\Projects\\Visual](file:///\\ToolsOptionsPages\Projects\Visual) D Settings\\DMD Directories"w;

static const wstring regPathToolsDirsGdc = "[\\ToolsOptionsPages\\Projects\\Visual](file:///\\ToolsOptionsPages\Projects\Visual) D Settings\\GDC Directories"w;

static const wstring regPathToolsDirsLdc = "[\\ToolsOptionsPages\\Projects\\Visual](file:///\\ToolsOptionsPages\Projects\Visual) D Settings\\LDC Directories"w;

static const wstring regMiscFiles = regPathProjects ~ "\\{A2FE74E1-B743-11d0-AE1A-00A0C90FFFC3}"w;

static const wstring regPathMetricsExcpt = "[\\AD7Metrics\\Exception"w](file:///\\AD7Metrics\Exception%22w);

static const wstring regPathMetricsEE = "[\\AD7Metrics\\ExpressionEvaluator"w](file:///\\AD7Metrics\ExpressionEvaluator%22w);

static const wstring vendorMicrosoftGuid = "{994B45C4-E6E9-11D2-903F-00C04FA302A1}"w;

static const wstring guidCOMPlusNativeEng = "{92EF0900-2251-11D2-B72E-0000F87572EF}"w;

///////////////////////////////////////////////////////////////////////

// Registration

///////////////////////////////////////////////////////////////////////

wstring GetRegistrationRoot(in wchar\* pszRegRoot, bool useRanu)

{

wstring szRegistrationRoot;

// figure out registration root, append "Configuration" in the case of RANU

if (pszRegRoot is null)

szRegistrationRoot = regPathConfigDefault;

else

szRegistrationRoot = to\_wstring(pszRegRoot);

if(useRanu)

{

scope RegKey keyConfig = new RegKey(HKEY\_CURRENT\_USER, szRegistrationRoot ~ "\_Config"w, false);

if(keyConfig.key)

szRegistrationRoot ~= "\_Config"w; // VS2010

else

szRegistrationRoot ~= "\\Configuration"w;

}

return szRegistrationRoot;

}

float guessVSVersion(wstring registrationRoot)

{

auto idx = lastIndexOf(registrationRoot, '\\');

if(idx < 0)

return 0;

wstring txt = registrationRoot[idx + 1 .. $];

return parse!float(txt);

}

void updateConfigurationChanged(HKEY keyRoot, wstring registrationRoot)

{

float ver = guessVSVersion(registrationRoot);

//MessageBoxA(null, text("version: ", ver, "\nregkey: ", to!string(registrationRoot)).ptr, to!string(registrationRoot).ptr, MB\_OK);

if(ver >= 11)

{

scope RegKey keyRegRoot = new RegKey(keyRoot, registrationRoot, true, false);

FILETIME fileTime;

GetSystemTimeAsFileTime(&fileTime);

ULARGE\_INTEGER ul;

ul.HighPart = fileTime.dwHighDateTime;

ul.LowPart = fileTime.dwLowDateTime;

ulong tempHNSecs = ul.QuadPart;

keyRegRoot.Set("ConfigurationChanged", tempHNSecs);

}

}

void fixVS2012Shellx64Debugger(HKEY keyRoot, wstring registrationRoot)

{

float ver = guessVSVersion(registrationRoot);

//MessageBoxA(null, text("version: ", ver, "\nregkey: ", to!string(registrationRoot)).ptr, to!string(registrationRoot).ptr, MB\_OK);

if(ver >= 11)

{

scope RegKey keyDebugger = new RegKey(keyRoot, registrationRoot ~ "\\Debugger"w);

keyDebugger.Set("msvsmon-pseudo\_remote"w, r"$ShellFolder$\Common7\Packages\Debugger\X64\msvsmon.exe"w, false);

}

}

HRESULT VSDllUnregisterServerInternal(in wchar\* pszRegRoot, in bool useRanu)

{

HKEY keyRoot = useRanu ? HKEY\_CURRENT\_USER : HKEY\_LOCAL\_MACHINE;

wstring registrationRoot = GetRegistrationRoot(pszRegRoot, useRanu);

wstring packageGuid = GUID2wstring(g\_packageCLSID);

wstring languageGuid = GUID2wstring(g\_languageCLSID);

wstring wizardGuid = GUID2wstring(g\_ProjectItemWizardCLSID);

wstring vdhelperGuid = GUID2wstring(g\_VisualDHelperCLSID);

HRESULT hr = S\_OK;

hr |= RegDeleteRecursive(keyRoot, registrationRoot ~ "[\\Packages\\"w](file:///\\Packages\%22w) ~ packageGuid);

hr |= RegDeleteRecursive(keyRoot, registrationRoot ~ "[\\CLSID\\"w](file:///\\CLSID\%22w) ~ languageGuid);

hr |= RegDeleteRecursive(keyRoot, registrationRoot ~ "[\\CLSID\\"w](file:///\\CLSID\%22w) ~ wizardGuid);

hr |= RegDeleteRecursive(keyRoot, registrationRoot ~ "[\\CLSID\\"w](file:///\\CLSID\%22w) ~ vdhelperGuid);

foreach (wstring fileExt; g\_languageFileExtensions)

hr |= RegDeleteRecursive(keyRoot, registrationRoot ~ regPathFileExts ~ "\\"w ~ fileExt);

hr |= RegDeleteRecursive(keyRoot, registrationRoot ~ "[\\Services\\"w](file:///\\Services\%22w) ~ languageGuid);

hr |= RegDeleteRecursive(keyRoot, registrationRoot ~ "[\\InstalledProducts\\"w](file:///\\InstalledProducts\%22w) ~ g\_packageName);

hr |= RegDeleteRecursive(keyRoot, registrationRoot ~ regPathLServices ~ "\\"w ~ g\_languageName);

hr |= RegDeleteRecursive(keyRoot, registrationRoot ~ regPathCodeExpansions ~ "\\"w ~ g\_languageName);

hr |= RegDeleteRecursive(keyRoot, registrationRoot ~ regPathPrjTemplates ~ "\\"w ~ packageGuid);

hr |= RegDeleteRecursive(keyRoot, registrationRoot ~ regPathProjects ~ "\\"w ~ GUID2wstring(g\_projectFactoryCLSID));

hr |= RegDeleteRecursive(keyRoot, registrationRoot ~ regMiscFiles ~ "[\\AddItemTemplates\\TemplateDirs\\"w](file:///\\AddItemTemplates\TemplateDirs\%22w) ~ packageGuid);

hr |= RegDeleteRecursive(keyRoot, registrationRoot ~ regPathToolsOptions);

foreach(guid; guids\_propertyPages)

hr |= RegDeleteRecursive(keyRoot, registrationRoot ~ "[\\CLSID\\"w](file:///\\CLSID\%22w) ~ GUID2wstring(\*guid));

hr |= RegDeleteRecursive(HKEY\_CLASSES\_ROOT, "CLSID\\"w ~ GUID2wstring(g\_unmarshalEnumOutCLSID));

static if(is(typeof(g\_unmarshalTargetInfoCLSID)))

hr |= RegDeleteRecursive(HKEY\_CLASSES\_ROOT, "CLSID\\"w ~ GUID2wstring(g\_unmarshalTargetInfoCLSID));

scope RegKey keyToolMenu = new RegKey(keyRoot, registrationRoot ~ "\\Menus"w);

keyToolMenu.Delete(packageGuid);

updateConfigurationChanged(keyRoot, registrationRoot);

return hr;

}

HRESULT VSDllRegisterServerInternal(in wchar\* pszRegRoot, in bool useRanu)

{

HKEY keyRoot = useRanu ? HKEY\_CURRENT\_USER : HKEY\_LOCAL\_MACHINE;

wstring registrationRoot = GetRegistrationRoot(pszRegRoot, useRanu);

wstring dllPath = GetDLLName(g\_hInst);

wstring templatePath = GetTemplatePath(dllPath);

wstring vdextPath = dirName(dllPath) ~ "\\vdextensions.dll"w;

try

{

wstring packageGuid = GUID2wstring(g\_packageCLSID);

wstring languageGuid = GUID2wstring(g\_languageCLSID);

wstring debugLangGuid = GUID2wstring(g\_debuggerLanguage);

wstring exprEvalGuid = GUID2wstring(g\_expressionEvaluator);

wstring wizardGuid = GUID2wstring(g\_ProjectItemWizardCLSID);

wstring vdhelperGuid = GUID2wstring(g\_VisualDHelperCLSID);

// package

scope RegKey keyPackage = new RegKey(keyRoot, registrationRoot ~ "[\\Packages\\"w](file:///\\Packages\%22w) ~ packageGuid);

keyPackage.Set(null, g\_packageName);

keyPackage.Set("InprocServer32"w, dllPath);

keyPackage.Set("About"w, g\_packageName);

keyPackage.Set("CompanyName"w, g\_packageCompany);

keyPackage.Set("ProductName"w, g\_packageName);

keyPackage.Set("ProductVersion"w, toUTF16(g\_packageVersion));

keyPackage.Set("MinEdition"w, "Standard");

keyPackage.Set("ID"w, 1);

int bspos = dllPath.length - 1;        while (bspos >= 0 && dllPath[bspos] != '\\') bspos--;

scope RegKey keySatellite = new RegKey(keyRoot, registrationRoot ~ "[\\Packages\\"w](file:///\\Packages\%22w) ~ packageGuid ~ "\\SatelliteDll"w);

keySatellite.Set("Path"w, dllPath[0 .. bspos+1]);

keySatellite.Set("DllName"w, ".."w ~ dllPath[bspos .. $]);

scope RegKey keyCLSID = new RegKey(keyRoot, registrationRoot ~ "[\\CLSID\\"w](file:///\\CLSID\%22w) ~ languageGuid);

keyCLSID.Set("InprocServer32"w, dllPath);

keyCLSID.Set("ThreadingModel"w, "Free"w); // Appartment?

// Wizards

scope RegKey keyWizardCLSID = new RegKey(keyRoot, registrationRoot ~ "[\\CLSID\\"w](file:///\\CLSID\%22w) ~ wizardGuid);

keyWizardCLSID.Set("InprocServer32"w, dllPath);

keyWizardCLSID.Set("ThreadingModel"w, "Appartment"w);

// VDExtensions

scope RegKey keyHelperCLSID = new RegKey(keyRoot, registrationRoot ~ "[\\CLSID\\"w](file:///\\CLSID\%22w) ~ vdhelperGuid);

keyHelperCLSID.Set("InprocServer32"w, "mscoree.dll");

keyHelperCLSID.Set("ThreadingModel"w, "Both"w);

keyHelperCLSID.Set(null, "vdextensions.VisualDHelper"w);

keyHelperCLSID.Set("Class"w, "vdextensions.VisualDHelper"w);

keyHelperCLSID.Set("CodeBase"w, vdextPath);

// file extensions

wstring fileExtensions;

foreach (wstring fileExt; g\_languageFileExtensions)

{

scope RegKey keyExt = new RegKey(keyRoot, registrationRoot ~ regPathFileExts ~ "\\"w ~ fileExt);

keyExt.Set(null, languageGuid);

keyExt.Set("Name"w, g\_languageName);

fileExtensions ~= fileExt ~ ";"w;

}

// language service

wstring langserv = registrationRoot ~ regPathLServices ~ "\\"w ~ g\_languageName;

scope RegKey keyLang = new RegKey(keyRoot, langserv);

keyLang.Set(null, languageGuid);

keyLang.Set("Package"w, packageGuid);

keyLang.Set("Extensions"w, fileExtensions);

keyLang.Set("LangResId"w, 0);

foreach (ref const(LanguageProperty) prop; g\_languageProperties)

keyLang.Set(prop.name, prop.value);

// colorizer settings

scope RegKey keyColorizer = new RegKey(keyRoot, langserv ~ "[\\EditorToolsOptions\\Colorizer"w](file:///\\EditorToolsOptions\Colorizer%22w));

keyColorizer.Set("Package"w, packageGuid);

keyColorizer.Set("Page"w, GUID2wstring(g\_ColorizerPropertyPage));

// intellisense settings

scope RegKey keyIntellisense = new RegKey(keyRoot, langserv ~ "[\\EditorToolsOptions\\Intellisense"w](file:///\\EditorToolsOptions\Intellisense%22w));

keyIntellisense.Set("Package"w, packageGuid);

keyIntellisense.Set("Page"w, GUID2wstring(g\_IntellisensePropertyPage));

scope RegKey keyService = new RegKey(keyRoot, registrationRoot ~ "[\\Services\\"w](file:///\\Services\%22w) ~ languageGuid);

keyService.Set(null, packageGuid);

keyService.Set("Name"w, g\_languageName);

scope RegKey keyProduct = new RegKey(keyRoot, registrationRoot ~ "[\\InstalledProducts\\"w](file:///\\InstalledProducts\%22w) ~ g\_packageName);

keyProduct.Set("Package"w, packageGuid);

keyProduct.Set("UseInterface"w, 1);

// snippets

wstring codeExp = registrationRoot ~ regPathCodeExpansions ~ "\\"w ~ g\_languageName;

scope RegKey keyCodeExp = new RegKey(keyRoot, codeExp);

keyCodeExp.Set(null, languageGuid);

keyCodeExp.Set("DisplayName"w, "131"w); // ???

keyCodeExp.Set("IndexPath"w, templatePath ~ "[\\CodeSnippets\\SnippetsIndex.xml"w](file:///\\CodeSnippets\SnippetsIndex.xml%22w));

keyCodeExp.Set("LangStringId"w, g\_languageName);

keyCodeExp.Set("Package"w, packageGuid);

keyCodeExp.Set("ShowRoots"w, 0);

wstring snippets = templatePath ~ "[\\CodeSnippets\\Snippets\\;%MyDocs%\\Code](file:///\\CodeSnippets\Snippets\;%25MyDocs%25\Code) Snippets\\" ~ g\_languageName ~ "[\\My](file:///\\My) Code Snippets\\"w;

scope RegKey keyCodeExp1 = new RegKey(keyRoot, codeExp ~ "\\ForceCreateDirs"w);

keyCodeExp1.Set(g\_languageName, snippets);

scope RegKey keyCodeExp2 = new RegKey(keyRoot, codeExp ~ "\\Paths"w);

keyCodeExp2.Set(g\_languageName, snippets);

scope RegKey keyPrjTempl = new RegKey(keyRoot, registrationRoot ~ regPathPrjTemplates ~ "\\"w ~ packageGuid ~ "[\\/1](file:///\\1)");

keyPrjTempl.Set(null, g\_languageName);

keyPrjTempl.Set("DeveloperActivity"w, g\_languageName);

keyPrjTempl.Set("SortPriority"w, 20);

keyPrjTempl.Set("TemplatesDir"w, templatePath ~ "\\Projects"w);

keyPrjTempl.Set("Folder"w, "{152CDB9D-B85A-4513-A171-245CE5C61FCC}"w); // other languages

// project

wstring projects = registrationRoot ~ "[\\Projects\\"w](file:///\\Projects\%22w) ~ GUID2wstring(g\_projectFactoryCLSID);

scope RegKey keyProject = new RegKey(keyRoot, projects);

keyProject.Set(null, "DProjectFactory"w);

keyProject.Set("DisplayName"w, g\_languageName);

keyProject.Set("DisplayProjectFileExtensions"w, g\_languageName ~ " Project Files (\*."w ~ g\_projectFileExtensions ~ ");\*."w ~ g\_projectFileExtensions);

keyProject.Set("Package"w, packageGuid);

keyProject.Set("DefaultProjectExtension"w, g\_projectFileExtensions);

keyProject.Set("PossibleProjectExtensions"w, g\_projectFileExtensions);

keyProject.Set("ProjectTemplatesDir"w, templatePath ~ "\\Projects"w);

keyProject.Set("Language(VsTemplate)"w, g\_languageName);

keyProject.Set("ItemTemplatesDir"w, templatePath ~ "\\Items"w);

// file templates

scope RegKey keyProject1 = new RegKey(keyRoot, projects ~ "[\\AddItemTemplates\\TemplateDirs\\"w](file:///\\AddItemTemplates\TemplateDirs\%22w) ~ packageGuid ~ "\\/1"w);

keyProject1.Set(null, g\_languageName);

keyProject1.Set("TemplatesDir"w, templatePath ~ "\\Items"w);

keyProject1.Set("SortPriority"w, 25);

// Miscellaneous Files Project

scope RegKey keyProject2 = new RegKey(keyRoot, registrationRoot ~ regMiscFiles ~ "[\\AddItemTemplates\\TemplateDirs\\"w](file:///\\AddItemTemplates\TemplateDirs\%22w) ~ packageGuid ~ "\\/1"w);

keyProject2.Set(null, g\_languageName);

keyProject2.Set("TemplatesDir"w, templatePath ~ "\\Items"w);

keyProject2.Set("SortPriority"w, 25);

// property pages

foreach(guid; guids\_propertyPages)

{

scope RegKey keyProp = new RegKey(keyRoot, registrationRoot ~ "[\\CLSID\\"w](file:///\\CLSID\%22w) ~ GUID2wstring(\*guid));

keyProp.Set("InprocServer32"w, dllPath);

keyProp.Set("ThreadingModel"w, "Appartment"w);

}

version(none){

// expression evaluator

scope RegKey keyLangDebug = new RegKey(keyRoot, langserv ~ "[\\Debugger](file:///\\Debugger) Languages\\"w ~ debugLangGuid);

keyLangDebug.Set(null, g\_languageName);

scope RegKey keyLangException = new RegKey(keyRoot, registrationRoot ~ regPathMetricsExcpt ~ "\\"w ~ debugLangGuid ~ "[\\D](file:///\\D) Exceptions");

wstring langEE = registrationRoot ~ regPathMetricsEE ~ "\\"w ~ debugLangGuid ~ "\\"w ~ vendorMicrosoftGuid;

scope RegKey keyLangEE = new RegKey(keyRoot, langEE);

keyLangEE.Set("CLSID"w, exprEvalGuid);

keyLangEE.Set("Language"w, g\_languageName);

keyLangEE.Set("Name"w, "D EE"w);

scope RegKey keyEngine = new RegKey(keyRoot, langEE ~ "[\\Engine](file:///\\Engine)");

keyEngine.Set("0"w, guidCOMPlusNativeEng);

}

// menu

scope RegKey keyToolMenu = new RegKey(keyRoot, registrationRoot ~ "\\Menus"w);

keyToolMenu.Set(packageGuid, ",2001,20"); // CTMENU,version

// Visual D settings

scope RegKey keyToolOpts = new RegKey(keyRoot, registrationRoot ~ regPathToolsOptions);

keyToolOpts.Set(null, "Visual D Settings");

keyToolOpts.Set("Package"w, packageGuid);

keyToolOpts.Set("Page"w, GUID2wstring(g\_ToolsProperty2Page));

// remove old page

RegDeleteRecursive(keyRoot, registrationRoot ~ regPathToolsDirsOld);

scope RegKey keyToolOptsDmd = new RegKey(keyRoot, registrationRoot ~ regPathToolsDirsDmd);

keyToolOptsDmd.Set(null, "DMD Directories");

keyToolOptsDmd.Set("Package"w, packageGuid);

keyToolOptsDmd.Set("Page"w, GUID2wstring(g\_DmdDirPropertyPage));

scope RegKey keyToolOptsGdc = new RegKey(keyRoot, registrationRoot ~ regPathToolsDirsGdc);

keyToolOptsGdc.Set(null, "GDC Directories");

keyToolOptsGdc.Set("Package"w, packageGuid);

keyToolOptsGdc.Set("Page"w, GUID2wstring(g\_GdcDirPropertyPage));

scope RegKey keyToolOptsLdc = new RegKey(keyRoot, registrationRoot ~ regPathToolsDirsLdc);

keyToolOptsLdc.Set(null, "LDC Directories");

keyToolOptsLdc.Set("Package"w, packageGuid);

keyToolOptsLdc.Set("Page"w, GUID2wstring(g\_LdcDirPropertyPage));

// remove "SkipLoading" entry from user settings

scope RegKey userKeyPackage = new RegKey(HKEY\_CURRENT\_USER, registrationRoot ~ "[\\Packages\\"w](file:///\\Packages\%22w) ~ packageGuid);

userKeyPackage.Delete("SkipLoading");

// remove Text Editor FontsAndColors Cache to add new Colors provided by Visual D

RegDeleteRecursive(HKEY\_CURRENT\_USER, registrationRoot ~ "[\\FontAndColors\\Cache](file:///\\FontAndColors\Cache)"); // \\{A27B4E24-A735-4D1D-B8E7-9716E1E3D8E0}");

// global registry keys for marshalled objects

void registerMarshalObject(ref in GUID iid)

{

scope RegKey keyMarshal1 = new RegKey(HKEY\_CLASSES\_ROOT, "CLSID\\"w ~ GUID2wstring(iid) ~ "\\InprocServer32"w);

keyMarshal1.Set(null, dllPath);

keyMarshal1.Set("ThreadingModel"w, "Both"w);

scope RegKey keyMarshal2 = new RegKey(HKEY\_CLASSES\_ROOT, "CLSID\\"w ~ GUID2wstring(iid) ~ "\\InprocHandler32"w);

keyMarshal2.Set(null, dllPath);

}

registerMarshalObject(g\_unmarshalEnumOutCLSID);

static if(is(typeof(g\_unmarshalTargetInfoCLSID)))

registerMarshalObject(g\_unmarshalTargetInfoCLSID);

fixVS2012Shellx64Debugger(keyRoot, registrationRoot);

updateConfigurationChanged(keyRoot, registrationRoot);

}

catch(RegistryException e)

{

return e.result;

}

return S\_OK;

}

wstring GetDLLName(HINSTANCE inst)

{

//get dll path

wchar[MAX\_PATH+1] dllPath;

DWORD dwLen = GetModuleFileNameW(inst, dllPath.ptr, MAX\_PATH);

if (dwLen == 0)

throw new RegistryException(HRESULT\_FROM\_WIN32(GetLastError()));

if (dwLen == MAX\_PATH)

throw new RegistryException(HRESULT\_FROM\_WIN32(ERROR\_INSUFFICIENT\_BUFFER));

return to\_wstring(dllPath.ptr);

}

wstring GetTemplatePath(wstring dllpath)

{

string path = toUTF8(dllpath);

path = dirName(path);

debug path = dirName(dirName(path)) ~ "[\\visuald](file:///\\visuald)";

path = path ~ "[\\Templates](file:///\\Templates)";

return toUTF16(path);

}

// This file is part of Visual D

//

// Visual D integrates the D programming language into Visual Studio

// Copyright (c) 2010 by Rainer Schuetze, All Rights Reserved

//

// Distributed under the Boost Software License, Version 1.0.

// See accompanying file LICENSE\_1\_0.txt or copy at <http://www.boost.org/LICENSE_1_0.txt>

module visuald.searchsymbol;

import visuald.windows;

import visuald.winctrl;

import visuald.comutil;

import visuald.hierutil;

import visuald.logutil;

import visuald.stringutil;

import visuald.fileutil;

import visuald.wmmsg;

import visuald.register;

import visuald.dpackage;

import visuald.intellisense;

import visuald.dimagelist;

import sdk.win32.commctrl;

import sdk.vsi.vsshell;

import sdk.vsi.vsshell80;

import dte80a = sdk.vsi.dte80a;

import dte80 = sdk.vsi.dte80;

import stdext.path;

import stdext.string;

import std.utf;

import std.algorithm;

import std.datetime;

import std.math;

import std.string;

import std.path;

import std.file;

import std.conv;

import std.exception;

import std.array;

import core.stdc.stdio : sprintf;

private IVsWindowFrame sWindowFrame;

private        SearchPane sSearchPane;

SearchPane getSearchPane(bool create)

{

if(!sSearchPane && create)

sSearchPane = newCom!SearchPane;

return sSearchPane;

}

bool showSearchWindow()

{

if(!getSearchPane(true))

return false;

if(!sWindowFrame)

{

auto pIVsUIShell = ComPtr!(IVsUIShell)(queryService!(IVsUIShell), false);

if(!pIVsUIShell)

return false;

const(wchar)\* caption = "Visual D Search"w.ptr;

HRESULT hr;

hr = pIVsUIShell.CreateToolWindow(CTW\_fInitNew, 0, sSearchPane,

&GUID\_NULL, &g\_searchWinCLSID, &GUID\_NULL,

null, caption, null, &sWindowFrame);

if(!SUCCEEDED(hr))

return false;

}

if(FAILED(sWindowFrame.Show()))

return false;

BOOL fHandled;

sSearchPane.\_OnSetFocus(0, 0, 0, fHandled);

return fHandled != 0;

}

bool showSearchWindow(bool searchFile, string word = "")

{

if(!showSearchWindow())

return false;

bool refresh = (sSearchPane.\_iqp.searchFile != searchFile);

if(refresh)

sSearchPane.\_ReinitViewState(searchFile, false);

if(!searchFile && word.length)

{

sSearchPane.\_iqp.wholeWord = true;

sSearchPane.\_iqp.caseSensitive = true;

sSearchPane.\_iqp.useRegExp = false;

refresh = true;

}

if(sSearchPane.\_wndFileWheel && word.length)

{

sSearchPane.\_wndFileWheel.SetWindowText(word);

refresh = true;

}

if(refresh)

sSearchPane.\_RefreshFileList();

return true;

}

bool closeSearchWindow()

{

sWindowFrame = release(sWindowFrame);

sSearchPane = null;

return true;

}

//const string kImageBmp = "imagebmp";

const int kColumnInfoVersion = 1;

const bool kToolBarAtTop = true;

const int kToolBarHeight = 24;

const int kPaneMargin = 0; // margin for back inside pane

const int kBackMargin = 2; // margin for controls inside back

struct static\_COLUMNINFO

{

string displayName;

int fmt;

int cx;

}

enum COLUMNID

{

NONE = -1,

NAME,

PATH,

SIZE,

LINE,

TYPE,

SCOPE,

MODIFIEDDATE,

KIND,

MAX

}

const static\_COLUMNINFO[] s\_rgColumns =

[

//{ "none", LVCFMT\_LEFT, 80 },

{ "Name", LVCFMT\_LEFT, 80 },

{ "Path", LVCFMT\_LEFT, 80 },

{ "Size", LVCFMT\_RIGHT, 80 },

{ "Line", LVCFMT\_RIGHT, 30 },

{ "Type", LVCFMT\_LEFT, 30 },

{ "Scope", LVCFMT\_LEFT, 80 },

{ "Date", LVCFMT\_LEFT, 80 },

{ "Kind", LVCFMT\_LEFT, 80 },

];

struct COLUMNINFO

{

COLUMNID colid;

BOOL fVisible;

int cx;

};

const COLUMNINFO[] default\_fileColumns =

[

{ COLUMNID.NAME, true, 100 },

{ COLUMNID.PATH, true, 200 },

{ COLUMNID.MODIFIEDDATE, true, 100 },

];

const COLUMNINFO[] default\_symbolColumns =

[

{ COLUMNID.NAME, true, 100 },

{ COLUMNID.TYPE, true, 50 },

{ COLUMNID.PATH, true, 200 },

{ COLUMNID.LINE, true, 50 },

{ COLUMNID.SCOPE, true, 100 },

{ COLUMNID.KIND, true, 100 },

];

struct INDEXQUERYPARAMS

{

COLUMNID colidSort;

bool fSortAscending;

COLUMNID colidGroup;

bool searchFile;

bool wholeWord;

bool caseSensitive;

bool useRegExp;

}

const HDMIL\_PRIVATE = 0xf00d;

class SearchWindowBack : Window

{

this(Window parent, SearchPane pane)

{

mPane = pane;

super(parent);

}

override int WindowProc(HWND hWnd, uint uMsg, WPARAM wParam, LPARAM lParam)

{

BOOL fHandled;

LRESULT rc = mPane.\_WindowProc(hWnd, uMsg, wParam, lParam, fHandled);

if(fHandled)

return rc;

return super.WindowProc(hWnd, uMsg, wParam, lParam);

}

SearchPane mPane;

}

class SearchPane : DisposingComObject, IVsWindowPane

{

static const GUID iid = uuid("FFA501E1-0565-4621-ADEA-9A8F10C1805B");

IServiceProvider mSite;

override HRESULT QueryInterface(in IID\* riid, void\*\* pvObject)

{

if(queryInterface!(SearchPane) (this, riid, pvObject))

return S\_OK;

if(queryInterface!(IVsWindowPane) (this, riid, pvObject))

return S\_OK;

// avoid debug output

if(\*riid == IVsCodeWindow.iid || \*riid == IServiceProvider.iid || \*riid == IVsTextView.iid)

return E\_NOINTERFACE;

return super.QueryInterface(riid, pvObject);

}

override void Dispose()

{

mSite = release(mSite);

}

HRESULT SetSite(/+[in]+/ IServiceProvider pSP)

{

mixin(LogCallMix2);

mSite = release(mSite);

mSite = addref(pSP);

return S\_OK;

}

HRESULT CreatePaneWindow(in HWND hwndParent, in int x, in int y, in int cx, in int cy,

/+[out]+/ HWND \*hwnd)

{

mixin(LogCallMix2);

\_wndParent = new Window(hwndParent);

\_wndBack = new SearchWindowBack(\_wndParent, this);

BOOL fHandled;

\_OnInitDialog(WM\_INITDIALOG, 0, 0, fHandled);

\_CheckSize();

\_wndBack.setVisible(true);

return S\_OK;

}

HRESULT GetDefaultSize(/+[out]+/ SIZE \*psize)

{

mixin(LogCallMix2);

psize.cx = 300;

psize.cy = 200;

return S\_OK;

}

HRESULT ClosePane()

{

mixin(LogCallMix2);

if(\_wndParent)

{

\_WriteViewStateToRegistry();

\_wndParent.Dispose();

\_wndParent = null;

\_wndBack = null;

\_wndFileWheel = null;

\_wndFileList = null;

\_wndFileListHdr = null;

\_wndToolbar = null;

if(\_himlToolbar)

ImageList\_Destroy(\_himlToolbar);

\_lastResultsArray = null;

mDlgFont = deleteDialogFont(mDlgFont);

}

return S\_OK;

}

HRESULT LoadViewState(/+[in]+/ IStream pstream)

{

mixin(LogCallMix2);

if(!pstream)

return E\_INVALIDARG;

HRESULT \_doRead(void\* p, size\_t cnt)

{

uint read;

HRESULT hr = pstream.Read(cast(byte\*)p, cnt, &read);

if(FAILED(hr))

return hr;

if(read != cnt)

return E\_UNEXPECTED;

return hr;

}

HRESULT \_doReadColumn(ref COLUMNINFO[] columns)

{

uint num;

if(HRESULT hr = \_doRead(cast(byte\*)&num, num.sizeof))

return hr;

if(num > 10)

return E\_UNEXPECTED;

columns.length = num;

if(HRESULT hr = \_doRead(columns.ptr, columns.length \* COLUMNINFO.sizeof))

return hr;

return S\_OK;

}

uint size;

if(HRESULT hr = \_doRead(cast(byte\*)&size, size.sizeof))

return hr;

if(HRESULT hr = \_doReadColumn(\_fileColumns))

return hr;

if(HRESULT hr = \_doReadColumn(\_symbolColumns))

return hr;

return S\_OK;

}

HRESULT SaveViewState(/+[in]+/ IStream pstream)

{

mixin(LogCallMix2);

if(!pstream)

return E\_INVALIDARG;

HRESULT \_doWrite(const(void)\* p, size\_t cnt)

{

uint written;

HRESULT hr = pstream.Write(cast(const(byte)\*)p, cnt, &written);

if(FAILED(hr))

return hr;

if(written != cnt)

return E\_UNEXPECTED;

return hr;

}

HRESULT \_doWriteColumn(COLUMNINFO[] columns)

{

uint num = columns.length;

if(HRESULT hr = \_doWrite(cast(byte\*)&num, num.sizeof))

return hr;

if(HRESULT hr = \_doWrite(columns.ptr, columns.length \* COLUMNINFO.sizeof))

return hr;

return S\_OK;

}

// write size overall to allow skipping chunk

uint size = 2 \* uint.sizeof + (\_fileColumns.length + \_symbolColumns.length) \* COLUMNINFO.sizeof;

if(HRESULT hr = \_doWrite(cast(byte\*)&size, size.sizeof))

return hr;

if(HRESULT hr = \_doWriteColumn(\_fileColumns))

return hr;

if(HRESULT hr = \_doWriteColumn(\_symbolColumns))

return hr;

return S\_OK;

}

HRESULT TranslateAccelerator(MSG\* msg)

{

if(msg.message == WM\_TIMER)

\_CheckSize();

if(msg.message == WM\_TIMER || msg.message == WM\_SYSTIMER)

return E\_NOTIMPL; // do not flood debug output

logMessage("TranslateAccelerator", msg.hwnd, msg.message, msg.wParam, msg.lParam);

BOOL fHandled;

HRESULT hrRet = \_HandleMessage(msg.hwnd, msg.message, msg.wParam, msg.lParam, fHandled);

if(fHandled)

return hrRet;

return E\_NOTIMPL;

}

///////////////////////////////////////////////////////////////////

// the following has been ported from the FlatSolutionExplorer project

private:

SolutionItemIndex \_spsii;

// DWORD \_dwIndexEventsCookie;

Window \_wndParent;

SearchWindowBack \_wndBack;

Text \_wndFileWheel;

ListView \_wndFileList;

Window \_wndFileListHdr;

ToolBar \_wndToolbar;

HIMAGELIST \_himlToolbar;

ItemArray \_lastResultsArray; // remember to keep reference to SolutionItems referenced in list items

HFONT mDlgFont;

BOOL \_fCombineColumns;

BOOL \_fAlternateRowColor;

BOOL \_closeOnReturn;

COLUMNINFO[] \_fileColumns;

COLUMNINFO[] \_symbolColumns;

COLUMNINFO[]\* \_rgColumns;

INDEXQUERYPARAMS \_iqp;

COLORREF \_crAlternate;

static HINSTANCE getInstance() { return Widget.getInstance(); }

int \_WindowProc(HWND hWnd, uint uMsg, WPARAM wParam, LPARAM lParam, ref BOOL fHandled)

{

if(uMsg != WM\_NOTIFY)

logMessage("\_WindowProc", hWnd, uMsg, wParam, lParam);

return \_HandleMessage(hWnd, uMsg, wParam, lParam, fHandled);

}

int \_HandleMessage(HWND hWnd, uint uMsg, WPARAM wParam, LPARAM lParam, ref BOOL fHandled)

{

switch(uMsg)

{

case WM\_CREATE:

case WM\_INITDIALOG:

return \_OnInitDialog(uMsg, wParam, lParam, fHandled);

case WM\_NCCALCSIZE:

return \_OnCalcSize(uMsg, wParam, lParam, fHandled);

case WM\_SIZE:

return \_OnSize(uMsg, wParam, lParam, fHandled);

case WM\_NCACTIVATE:

case WM\_SETFOCUS:

return \_OnSetFocus(uMsg, wParam, lParam, fHandled);

case WM\_CONTEXTMENU:

return \_OnContextMenu(uMsg, wParam, lParam, fHandled);

case WM\_DESTROY:

return \_OnDestroy(uMsg, wParam, lParam, fHandled);

case WM\_KEYDOWN:

case WM\_SYSKEYDOWN:

return \_OnKeyDown(uMsg, wParam, lParam, fHandled);

case WM\_COMMAND:

ushort id = LOWORD(wParam);

ushort code = HIWORD(wParam);

if(id == IDC\_FILEWHEEL && code == EN\_CHANGE)

return \_OnFileWheelChanged(id, code, hWnd, fHandled);

if(code == BN\_CLICKED)

{

switch(id)

{

case IDOK:

return \_OnOpenSelectedItem(code, id, hWnd, fHandled);

case IDR\_COMBINECOLUMNS:

case IDR\_ALTERNATEROWCOLOR:

case IDR\_GROUPBYKIND:

case IDR\_CLOSEONRETURN:

case IDR\_WHOLEWORD:

case IDR\_CASESENSITIVE:

case IDR\_REGEXP:

case IDR\_SEARCHFILE:

case IDR\_SEARCHSYMBOL:

return \_OnCheckBtnClicked(code, id, hWnd, fHandled);

default:

break;

}

}

break;

case WM\_NOTIFY:

NMHDR\* nmhdr = cast(NMHDR\*)lParam;

if(nmhdr.idFrom == IDC\_FILELIST)

{

switch(nmhdr.code)

{

case LVN\_GETDISPINFO:

return \_OnFileListGetDispInfo(wParam, nmhdr, fHandled);

case LVN\_COLUMNCLICK:

return \_OnFileListColumnClick(wParam, nmhdr, fHandled);

case LVN\_DELETEITEM:

return \_OnFileListDeleteItem(wParam, nmhdr, fHandled);

case NM\_DBLCLK:

return \_OnFileListDblClick(wParam, nmhdr, fHandled);

case NM\_CUSTOMDRAW:

return \_OnFileListCustomDraw(wParam, nmhdr, fHandled);

default:

break;

}

}

if (nmhdr.idFrom == IDC\_FILELISTHDR && nmhdr.code == HDN\_ITEMCHANGED)

return \_OnFileListHdrItemChanged(wParam, nmhdr, fHandled);

if (nmhdr.idFrom == IDC\_TOOLBAR && nmhdr.code == TBN\_GETINFOTIP)

return \_OnToolbarGetInfoTip(wParam, nmhdr, fHandled);

break;

default:

break;

}

return 0;

}

public this()

{

\_fAlternateRowColor = true;

\_closeOnReturn = true;

\_spsii = new SolutionItemIndex();

\_fileColumns = default\_fileColumns.dup;

\_symbolColumns = default\_symbolColumns.dup;

\_iqp.colidSort = COLUMNID.NAME;

\_iqp.fSortAscending = true;

\_iqp.colidGroup = COLUMNID.NONE;

\_rgColumns = \_iqp.searchFile ? &\_fileColumns : &\_symbolColumns;

}

void \_MoveSelection(BOOL fDown)

{

// Get the current selection

int iSel = \_wndFileList.SendMessage(LVM\_GETNEXTITEM, cast(WPARAM)-1, LVNI\_SELECTED);

int iCnt = \_wndFileList.SendMessage(LVM\_GETITEMCOUNT);

if(iSel == 0 && !fDown)

return;

if(iSel == iCnt - 1 && fDown)

return;

LVITEM lvi;

lvi.iItem = iSel; // fDown ? iSel+1 : iSel-1;

lvi.mask = LVIF\_STATE;

lvi.stateMask = LVIS\_SELECTED | LVIS\_FOCUSED;

lvi.state = 0;

\_wndFileList.SendItemMessage(LVM\_SETITEM, lvi);

lvi.iItem = fDown ? iSel+1 : iSel-1;

lvi.mask = LVIF\_STATE;

lvi.stateMask = LVIS\_SELECTED | LVIS\_FOCUSED;

lvi.state = LVIS\_SELECTED | LVIS\_FOCUSED;

\_wndFileList.SendItemMessage(LVM\_SETITEM, lvi);

\_wndFileList.SendMessage(LVM\_ENSUREVISIBLE, lvi.iItem, FALSE);

}

HRESULT \_PrepareFileListForResults(in ItemArray puaResults)

{

\_wndFileList.SendMessage(LVM\_DELETEALLITEMS);

\_wndFileList.SendMessage(LVM\_REMOVEALLGROUPS);

HIMAGELIST himl = LoadImageList(getInstance(), MAKEINTRESOURCEA(BMP\_DIMAGELIST), 16, 16);

if(himl)

\_wndFileList.SendMessage(LVM\_SETIMAGELIST, LVSIL\_SMALL, cast(LPARAM)himl);

HRESULT hr = S\_OK;

BOOL fEnableGroups = \_iqp.colidGroup != COLUMNID.NONE;

if (fEnableGroups)

{

DWORD cGroups = puaResults.GetCount();

// Don't enable groups if there is only 1

if (cGroups <= 1)

{

fEnableGroups = FALSE;

}

}

if (SUCCEEDED(hr))

{

hr = \_wndFileList.SendMessage(LVM\_ENABLEGROUPVIEW, fEnableGroups) == -1 ? E\_FAIL : S\_OK;

}

return hr;

}

HRESULT \_AddItemsToFileList(int iGroupId, in ItemArray pua)

{

LVITEM lvi;

lvi.pszText = LPSTR\_TEXTCALLBACK;

lvi.iItem = cast(int)\_wndFileList.SendMessage(LVM\_GETITEMCOUNT);

DWORD cItems = pua.GetCount();

HRESULT hr = S\_OK;

for (DWORD i = 0; i < cItems && SUCCEEDED(hr); i++)

{

if(SolutionItem spsi = pua.GetItem(i))

{

for (int iCol = COLUMNID.NAME; iCol < COLUMNID.MAX; iCol++)

{

lvi.iSubItem = iCol;

if (iCol != COLUMNID.NAME)

{

lvi.mask = LVIF\_TEXT;

}

else

{

lvi.mask = LVIF\_PARAM | LVIF\_TEXT | LVIF\_IMAGE;

lvi.iGroupId = iGroupId;

lvi.lParam = cast(LPARAM)cast(void\*)spsi;

lvi.iImage = spsi.GetIconIndex();

if (iGroupId != -1)

{

lvi.mask |= LVIF\_GROUPID;

lvi.iGroupId = iGroupId;

}

}

if (\_wndFileList.SendItemMessage(LVM\_INSERTITEM, lvi) != -1 && iCol == COLUMNID.NAME)

{

//spsi.detach();

}

}

spsi = null;

}

lvi.iItem++;

}

return hr;

}

HRESULT \_AddGroupToFileList(int iGroupId, in SolutionItemGroup psig)

{

LVGROUP lvg;

lvg.cbSize = lvg.sizeof;

lvg.mask = LVGF\_ALIGN | LVGF\_HEADER | LVGF\_GROUPID | LVGF\_STATE;

lvg.uAlign = LVGA\_HEADER\_LEFT;

lvg.iGroupId = iGroupId;

lvg.pszHeader = \_toUTF16z(psig.GetName());

lvg.state = LVGS\_NORMAL;

HRESULT hr = \_wndFileList.SendMessage(LVM\_INSERTGROUP, cast(WPARAM)-1, cast(LPARAM)&lvg) != -1 ? S\_OK : E\_FAIL;

if (SUCCEEDED(hr))

{

const(ItemArray) spItems = psig.GetItems();

if(spItems)

{

hr = \_AddItemsToFileList(iGroupId, spItems);

}

}

return hr;

}

HRESULT \_RefreshFileList()

{

mixin(LogCallMix);

\_wndFileList.SetRedraw(FALSE);

HRESULT hr = S\_OK;

string strWordWheel = \_wndFileWheel.GetWindowText();

ItemArray spResultsArray;

hr = \_spsii.Search(strWordWheel, &\_iqp, &spResultsArray);

if (SUCCEEDED(hr))

{

hr = \_PrepareFileListForResults(spResultsArray);

if (SUCCEEDED(hr))

{

if (\_iqp.colidGroup != COLUMNID.NONE)

{

DWORD cGroups = spResultsArray.GetCount();

for (DWORD iGroup = 0; iGroup < cGroups && SUCCEEDED(hr); iGroup++)

{

if(SolutionItemGroup spsig = spResultsArray.GetGroup(iGroup))

{

hr = \_AddGroupToFileList(iGroup, spsig);

}

}

}

else

{

hr = \_AddItemsToFileList(-1, spResultsArray);

}

}

\_lastResultsArray = spResultsArray;

}

if (SUCCEEDED(hr))

{

// Select the first item

LVITEM lviSelect;

lviSelect.mask = LVIF\_STATE;

lviSelect.iItem = 0;

lviSelect.state = LVIS\_SELECTED | LVIS\_FOCUSED;

lviSelect.stateMask = LVIS\_SELECTED | LVIS\_FOCUSED;

\_wndFileList.SendItemMessage(LVM\_SETITEM, lviSelect);

}

\_wndFileList.SetRedraw(TRUE);

\_wndFileList.InvalidateRect(null, FALSE);

return hr;

}

// Special icon dimensions for the sort direction indicator

enum int c\_cxSortIcon = 7;

enum int c\_cySortIcon = 6;

HRESULT \_CreateSortImageList(out HIMAGELIST phiml)

{

// Create an image list for the sort direction indicators

HIMAGELIST himl = ImageList\_Create(c\_cxSortIcon, c\_cySortIcon, ILC\_COLORDDB | ILC\_MASK, 2, 1);

HRESULT hr = himl ? S\_OK : E\_OUTOFMEMORY;

if (SUCCEEDED(hr))

{

HICON hicn = cast(HICON)LoadImage(getInstance(), MAKEINTRESOURCE(IDI\_DESCENDING), IMAGE\_ICON, c\_cxSortIcon, c\_cySortIcon, LR\_DEFAULTCOLOR | LR\_SHARED);

hr = hicn ? S\_OK : HResultFromLastError();

if (SUCCEEDED(hr))

{

hr = ImageList\_ReplaceIcon(himl, -1, hicn) != -1 ? S\_OK : E\_FAIL;

if (SUCCEEDED(hr))

{

hicn = cast(HICON)LoadImage(getInstance(), MAKEINTRESOURCE(IDI\_ASCENDING), IMAGE\_ICON, c\_cxSortIcon, c\_cySortIcon, LR\_DEFAULTCOLOR | LR\_SHARED);

hr = hicn ? S\_OK : HResultFromLastError();

if (SUCCEEDED(hr))

{

hr = ImageList\_ReplaceIcon(himl, -1, hicn) != -1 ? S\_OK : E\_FAIL;

if (SUCCEEDED(hr))

{

phiml = himl;

himl = null;

}

}

}

}

if (himl)

{

ImageList\_Destroy(himl);

}

}

return hr;

}

HRESULT \_AddSortIcon(int iIndex, BOOL fAscending)

{

if(iIndex < 0)

return E\_FAIL;

// First, get the current header item fmt

HDITEM hdi;

hdi.mask = HDI\_FORMAT;

HRESULT hr = \_wndFileListHdr.SendMessage(HDM\_GETITEM, iIndex, cast(LPARAM)&hdi) ? S\_OK : E\_FAIL;

if (SUCCEEDED(hr))

{

// Add the image mask and alignment

hdi.mask |= HDI\_IMAGE;

hdi.fmt |= HDF\_IMAGE;

if ((hdi.fmt & HDF\_JUSTIFYMASK) == HDF\_LEFT)

{

hdi.fmt |= HDF\_BITMAP\_ON\_RIGHT;

}

hdi.iImage = fAscending;

hr = \_wndFileListHdr.SendMessage(HDM\_SETITEM, iIndex, cast(LPARAM)&hdi) ? S\_OK : E\_FAIL;

}

return hr;

}

HRESULT \_RemoveSortIcon(int iIndex)

{

if(iIndex < 0)

return E\_FAIL;

// First, get the current header item fmt

HDITEM hdi;

hdi.mask = HDI\_FORMAT;

HRESULT hr = \_wndFileListHdr.SendMessage(HDM\_GETITEM, iIndex, cast(LPARAM)&hdi) ? S\_OK : E\_FAIL;

if (SUCCEEDED(hr))

{

// Remove the image mask and alignment

hdi.fmt &= ~HDF\_IMAGE;

if ((hdi.fmt & HDF\_JUSTIFYMASK) == HDF\_LEFT)

{

hdi.fmt &= ~HDF\_BITMAP\_ON\_RIGHT;

}

hr = \_wndFileListHdr.SendMessage(HDM\_SETITEM, iIndex, cast(LPARAM)&hdi) ? S\_OK : E\_FAIL;

}

return hr;

}

HRESULT \_InsertListViewColumn(int iIndex, COLUMNID colid, int cx, bool set = false)

{

LVCOLUMN lvc;

lvc.mask = LVCF\_FMT | LVCF\_TEXT | LVCF\_WIDTH;

lvc.fmt = s\_rgColumns[colid].fmt;

lvc.cx = cx;

HRESULT hr = S\_OK;

string strDisplayName = s\_rgColumns[colid].displayName;

lvc.pszText = \_toUTF16z(strDisplayName);

uint msg = set ? LVM\_SETCOLUMNW : LVM\_INSERTCOLUMNW;

hr = \_wndFileList.SendMessage(msg, iIndex, cast(LPARAM)&lvc) >= 0 ? S\_OK : E\_FAIL;

if (SUCCEEDED(hr))

{

HDITEM hdi;

hdi.mask = HDI\_LPARAM;

hdi.lParam = colid;

hr = \_wndFileListHdr.SendMessage(HDM\_SETITEM, iIndex, cast(LPARAM)&hdi) ? S\_OK : E\_FAIL;

}

return hr;

}

HRESULT \_InitializeFileListColumns()

{

\_wndFileList.SendMessage(LVM\_DELETEALLITEMS);

\_wndFileList.SendMessage(LVM\_REMOVEALLGROUPS);

bool hasNameColumn = \_wndFileList.SendMessage(LVM\_GETCOLUMNWIDTH, 0) > 0;

// cannot delete col 0, so keep name

while(\_wndFileList.SendMessage(LVM\_DELETECOLUMN, 1)) {}

HRESULT hr = S\_OK;

COLUMNID colPath = \_iqp.searchFile ? COLUMNID.PATH : COLUMNID.TYPE;

int cColumnsInserted = 0;

for (UINT i = 0; i < \_rgColumns.length && SUCCEEDED(hr); i++)

{

COLUMNINFO\* ci = &(\*\_rgColumns)[i];

if (ci.fVisible)

{

// Don't insert the path column if we're compressing path and filename

if (ci.colid != colPath || !\_fCombineColumns)

{

int cx = ci.cx;

if (ci.colid == COLUMNID.NAME && \_fCombineColumns)

{

COLUMNINFO \*pci = \_ColumnInfoFromColumnID(colPath);

cx += pci.cx;

}

bool set = hasNameColumn ? cColumnsInserted == 0 : false;

hr = \_InsertListViewColumn(cColumnsInserted++, ci.colid, cx, set);

}

}

}

return hr;

}

HRESULT \_InitializeFileList()

{

\_wndFileList.SendMessage(LVM\_SETEXTENDEDLISTVIEWSTYLE,

LVS\_EX\_FULLROWSELECT | LVS\_EX\_DOUBLEBUFFER | LVS\_EX\_LABELTIP,

LVS\_EX\_FULLROWSELECT | LVS\_EX\_DOUBLEBUFFER | LVS\_EX\_LABELTIP);

HIMAGELIST himl;

HRESULT hr = \_CreateSortImageList(himl);

if (SUCCEEDED(hr))

{

\_wndFileListHdr.SendMessage(HDM\_SETIMAGELIST, HDMIL\_PRIVATE, cast(LPARAM)himl);

\_InitializeFileListColumns();

if (SUCCEEDED(hr))

{

hr = \_AddSortIcon(\_ListViewIndexFromColumnID(\_iqp.colidSort), \_iqp.fSortAscending);

if (SUCCEEDED(hr))

{

\_RefreshFileList();

}

}

}

return hr;

}

// Special icon dimensions for the toolbar images

enum int c\_cxToolbarIcon = 16;

enum int c\_cyToolbarIcon = 15;

HRESULT \_CreateToolbarImageList(out HIMAGELIST phiml)

{

// Create an image list for the sort direction indicators

int icons = IDR\_LAST - IDR\_FIRST + 1;

HIMAGELIST himl = ImageList\_Create(c\_cxToolbarIcon, c\_cyToolbarIcon, ILC\_COLORDDB | ILC\_MASK, icons, 1);

HRESULT hr = himl ? S\_OK : E\_OUTOFMEMORY;

if (SUCCEEDED(hr))

{

// icons have image index IDR\_XXX - IDR\_FIRST

for (int i = IDR\_FIRST; i <= IDR\_LAST && SUCCEEDED(hr); i++)

{

HICON hicn = cast(HICON)LoadImage(getInstance(), MAKEINTRESOURCE(i),

IMAGE\_ICON, c\_cxToolbarIcon, c\_cyToolbarIcon, LR\_DEFAULTCOLOR | LR\_SHARED);

hr = hicn ? S\_OK : HResultFromLastError();

if (SUCCEEDED(hr))

{

hr = ImageList\_ReplaceIcon(himl, -1, hicn) != -1 ? S\_OK : E\_FAIL;

}

}

if (SUCCEEDED(hr))

{

phiml = himl;

himl = null;

}

if (himl)

{

ImageList\_Destroy(himl);

}

}

return hr;

}

HRESULT \_InitializeToolbar()

{

HRESULT hr = \_CreateToolbarImageList(\_himlToolbar);

if (SUCCEEDED(hr))

{

int style = CCS\_NODIVIDER | TBSTYLE\_FLAT | TBSTYLE\_TOOLTIPS | CCS\_NORESIZE;

//style |= (kToolBarAtTop ? CCS\_TOP : CCS\_BOTTOM);

\_wndToolbar = new ToolBar(\_wndBack, style, TBSTYLE\_EX\_DOUBLEBUFFER, IDC\_TOOLBAR);

hr = \_wndToolbar.hwnd ? S\_OK : E\_FAIL;

if (SUCCEEDED(hr))

{

\_wndToolbar.setRect(kBackMargin, kBackMargin, 100, kToolBarHeight);

\_wndToolbar.SendMessage(TB\_SETIMAGELIST, 0, cast(LPARAM)\_himlToolbar);

TBBUTTON btn2 = { 10, 11, TBSTATE\_ENABLED, 1, [0,0], 0, 0 };

TBBUTTON initButton(int id, ubyte style)

{

return TBBUTTON(id < 0 ? 10 : id - IDR\_FIRST, id, TBSTATE\_ENABLED, style, [0,0], 0, 0);

}

static const TBBUTTON[] s\_tbb = [

initButton(IDR\_SEARCHFILE, BTNS\_CHECKGROUP),

initButton(IDR\_SEARCHSYMBOL, BTNS\_CHECKGROUP),

initButton(-1, BTNS\_SEP),

initButton(IDR\_COMBINECOLUMNS, BTNS\_CHECK),

initButton(IDR\_ALTERNATEROWCOLOR, BTNS\_CHECK),

initButton(IDR\_GROUPBYKIND, BTNS\_CHECK),

initButton(IDR\_CLOSEONRETURN, BTNS\_CHECK),

initButton(-1, BTNS\_SEP),

initButton(IDR\_WHOLEWORD, BTNS\_CHECK),

initButton(IDR\_CASESENSITIVE, BTNS\_CHECK),

initButton(IDR\_REGEXP, BTNS\_CHECK),

];

hr = \_wndToolbar.SendMessage(TB\_ADDBUTTONS, s\_tbb.length, cast(LPARAM)s\_tbb.ptr) ? S\_OK : E\_FAIL;

if (SUCCEEDED(hr))

{

hr = \_InitializeSwitches();

}

}

}

return hr;

}

HRESULT \_InitializeSwitches()

{

// Set the initial state of the buttons

HRESULT hr = S\_OK;

\_wndToolbar.EnableCheckButton(IDR\_COMBINECOLUMNS, true, \_fCombineColumns != 0);

\_wndToolbar.EnableCheckButton(IDR\_ALTERNATEROWCOLOR, true, \_fAlternateRowColor != 0);

\_wndToolbar.EnableCheckButton(IDR\_CLOSEONRETURN, true, \_closeOnReturn != 0);

\_wndToolbar.EnableCheckButton(IDR\_GROUPBYKIND, true, \_iqp.colidGroup == COLUMNID.KIND);

\_wndToolbar.EnableCheckButton(IDR\_WHOLEWORD, true, \_iqp.wholeWord);

\_wndToolbar.EnableCheckButton(IDR\_CASESENSITIVE, true, !\_iqp.caseSensitive); // button on is case INsensitive

\_wndToolbar.EnableCheckButton(IDR\_REGEXP, true, \_iqp.useRegExp);

\_wndToolbar.EnableCheckButton(IDR\_SEARCHFILE, true, \_iqp.searchFile);

\_wndToolbar.EnableCheckButton(IDR\_SEARCHSYMBOL, true, !\_iqp.searchFile);

return hr;

}

extern(Windows) LRESULT \_HdrWndProc(HWND hwnd, UINT uiMsg, WPARAM wParam, LPARAM lParam)

{

LRESULT lRet = 0;

BOOL fHandled = FALSE;

switch (uiMsg)

{

case WM\_DESTROY:

RemoveWindowSubclass(hwnd, &s\_HdrWndProc, ID\_SUBCLASS\_HDR);

break;

case HDM\_SETIMAGELIST:

if (wParam == HDMIL\_PRIVATE)

{

wParam = 0;

}

else

{

fHandled = TRUE;

}

break;

default:

break;

}

if (!fHandled)

{

lRet = DefSubclassProc(hwnd, uiMsg, wParam, lParam);

}

return lRet;

}

static extern(Windows) LRESULT s\_HdrWndProc(HWND hWnd, UINT uiMsg, WPARAM wParam, LPARAM lParam, UINT\_PTR uIdSubclass, DWORD\_PTR dwRefData)

{

if(SearchPane pfsec = cast(SearchPane)cast(void\*)dwRefData)

return pfsec.\_HdrWndProc(hWnd, uiMsg, wParam, lParam);

return DefSubclassProc(hWnd, uiMsg, wParam, lParam);

}

LRESULT \_OnInitDialog(UINT uiMsg, WPARAM wParam, LPARAM lParam, ref BOOL fHandled)

{

if(\_wndFileWheel)

return S\_OK;

updateEnvironmentFont();

if(!mDlgFont)

mDlgFont = newDialogFont();

if (SUCCEEDED(\_InitializeViewState()))

{

\_wndFileWheel = new Text(\_wndBack, "", IDC\_FILEWHEEL);

int top = kToolBarAtTop ? kToolBarHeight : 1;

\_wndFileWheel.setRect(kBackMargin, top + 2 + kBackMargin, 185, 16);

\_wndFileList = new ListView(\_wndBack, LVS\_REPORT | LVS\_SINGLESEL | LVS\_SHOWSELALWAYS | LVS\_ALIGNLEFT | LVS\_SHAREIMAGELISTS | WS\_BORDER | WS\_TABSTOP,

0, IDC\_FILELIST);

\_wndFileList.setRect(kBackMargin, top + kBackMargin + 20, 185, 78);

HWND hdrHwnd = cast(HWND)\_wndFileList.SendMessage(LVM\_GETHEADER);

if(hdrHwnd)

{

\_wndFileListHdr = new Window(hdrHwnd);

// HACK: This header control is created by the listview. When listview handles LVM\_SETIMAGELIST with

// LVSIL\_SMALL it also forwards the message to the header control. The subclass proc will intercept those

// messages and prevent resetting the imagelist

SetWindowSubclass(\_wndFileListHdr.hwnd, &s\_HdrWndProc, ID\_SUBCLASS\_HDR, cast(DWORD\_PTR)cast(void\*)this);

//\_wndFileListHdr.SetDlgCtrlID(IDC\_FILELISTHDR);

}

\_InitializeFileList();

\_InitializeToolbar();

}

//return CComCompositeControl<CFlatSolutionExplorer>::OnInitDialog(uiMsg, wParam, lParam, fHandled);

return S\_OK;

}

LRESULT \_OnCalcSize(UINT uiMsg, WPARAM wParam, LPARAM lParam, ref BOOL fHandled)

{

//                \_CheckSize();

return 0;

}

void \_CheckSize()

{

RECT r, br;

\_wndParent.GetClientRect(&r);

\_wndBack.GetClientRect(&br);

if(br.right - br.left != r.right - r.left - 2\*kPaneMargin ||

br.bottom - br.top != r.bottom - r.top - 2\*kPaneMargin)

\_wndBack.setRect(kPaneMargin, kPaneMargin,

r.right - r.left - 2\*kPaneMargin, r.bottom - r.top - 2\*kPaneMargin);

}

LRESULT \_OnSize(UINT uiMsg, WPARAM wParam, LPARAM lParam, ref BOOL fHandled)

{

int cx = LOWORD(lParam);

int cy = HIWORD(lParam);

// Adjust child control sizes

// - File Wheel stretches to fit horizontally but size is vertically fixed

// - File List stretches to fit horizontally and vertically but the topleft coordinate is fixed

// - Toolbar autosizes along the bottom

\_wndToolbar.setRect(kBackMargin, kBackMargin, cx - 2 \* kBackMargin, kToolBarHeight);

RECT rcFileWheel;

if (\_wndFileWheel.GetWindowRect(&rcFileWheel))

{

\_wndBack.ScreenToClient(&rcFileWheel);

rcFileWheel.right = cx - kBackMargin;

\_wndFileWheel.SetWindowPos(null, &rcFileWheel, SWP\_NOMOVE | SWP\_NOZORDER | SWP\_NOACTIVATE);

RECT rcFileList;

if (\_wndFileList.GetWindowRect(&rcFileList))

{

\_wndBack.ScreenToClient(&rcFileList);

rcFileList.right = cx - kBackMargin;

rcFileList.bottom = cy - (kToolBarAtTop ? 0 : kToolBarHeight) - kBackMargin;

\_wndFileList.SetWindowPos(null, &rcFileList, SWP\_NOMOVE | SWP\_NOZORDER | SWP\_NOACTIVATE);

}

}

return 0;

}

LRESULT \_OnSetFocus(UINT uiMsg, WPARAM wParam, LPARAM lParam, ref BOOL fHandled)

{

// Skip the CComCompositeControl handling

// CComControl<CFlatSolutionExplorer, CAxDialogImpl<CFlatSolutionExplorer>>::OnSetFocus(uiMsg, wParam, lParam, fHandled);

if(\_wndFileWheel)

{

\_wndFileWheel.SetFocus();

\_wndFileWheel.SendMessage(EM\_SETSEL, 0, cast(LPARAM)-1);

fHandled = TRUE;

}

return 0;

}

LRESULT \_OnKeyDown(UINT uiMsg, WPARAM wParam, LPARAM lParam, ref BOOL fHandled)

{

//HWND hwndFocus = .GetFocus();

//UINT cItems = cast(UINT)\_wndFileList.SendMessage(LVM\_GETITEMCOUNT);

//if (cItems && hwndFocus == \_wndFileWheel.hwnd)

{

UINT vKey = LOWORD(wParam);

switch(vKey)

{

case VK\_UP:

case VK\_DOWN:

case VK\_PRIOR:

case VK\_NEXT:

fHandled = TRUE;

return \_wndFileList.SendMessage(uiMsg, wParam, lParam);

// \_MoveSelection(vKey == VK\_DOWN);

case VK\_RETURN:

case VK\_EXECUTE:

return \_OnOpenSelectedItem(0, 0, null, fHandled);

case VK\_ESCAPE:

if(\_closeOnReturn)

sWindowFrame.Hide();

break;

default:

break;

}

}

return 0;

}

HRESULT \_ToggleColumnVisibility(COLUMNID colid)

{

HRESULT hr = E\_FAIL;

COLUMNINFO \*pci = \_ColumnInfoFromColumnID(colid);

BOOL fVisible = !pci.fVisible;

if (fVisible)

{

int iIndex = 0;

BOOL fDone = FALSE;

COLUMNID colPath = \_iqp.searchFile ? COLUMNID.PATH : COLUMNID.TYPE;

for (size\_t i = 0; i < \_rgColumns.length && !fDone; i++)

{

COLUMNINFO \*ci = &(\*\_rgColumns)[i];

if (ci.colid == colid)

{

fDone = TRUE;

}

else if (ci.fVisible && (ci.colid != colPath || !\_fCombineColumns))

{

iIndex++;

}

}

hr = \_InsertListViewColumn(iIndex, colid, pci.cx);

if (SUCCEEDED(hr))

{

pci.fVisible = TRUE;

}

}

else

{

int iCol = \_ListViewIndexFromColumnID(colid);

hr = \_wndFileList.SendMessage(LVM\_DELETECOLUMN, iCol) ? S\_OK : E\_FAIL;

if (SUCCEEDED(hr))

{

pci.fVisible = fVisible;

if (colid == \_iqp.colidSort)

{

hr = \_SetSortColumn(COLUMNID.NAME, 0);

}

}

}

if (SUCCEEDED(hr))

{

\_WriteColumnInfoToRegistry();

}

return hr;

}

HRESULT \_ChooseColumns(POINT pt)

{

HMENU hmnu = CreatePopupMenu();

HRESULT hr = hmnu ? S\_OK : HResultFromLastError();

if (SUCCEEDED(hr))

{

MENUITEMINFO mii;

mii.cbSize = mii.sizeof;

mii.fMask = MIIM\_FTYPE | MIIM\_ID | MIIM\_STATE | MIIM\_STRING;

mii.fType = MFT\_STRING;

COLUMNID colPath = \_iqp.searchFile ? COLUMNID.PATH : COLUMNID.TYPE;

// Don't include the first column (COLUMNID.NAME) in the list

for (size\_t i = COLUMNID.NAME + 1; i < \_rgColumns.length && SUCCEEDED(hr); i++)

{

COLUMNINFO \*ci = &(\*\_rgColumns)[i];

string strDisplayName = s\_rgColumns[ci.colid].displayName;

mii.fState = (ci.colid == colPath && \_fCombineColumns) ? MFS\_DISABLED : MFS\_ENABLED;

if (ci.fVisible)

{

mii.fState |= MFS\_CHECKED;

}

mii.wID = ci.colid + IDM\_COLUMNLISTBASE;

mii.dwTypeData = \_toUTF16z(strDisplayName);

if(!InsertMenuItem(hmnu, cast(UINT)i-1, TRUE, &mii))

hr = HResultFromLastError();

}

if (SUCCEEDED(hr))

{

UINT uiCmd = TrackPopupMenuEx(hmnu, TPM\_RETURNCMD | TPM\_NONOTIFY | TPM\_HORIZONTAL | TPM\_TOPALIGN | TPM\_LEFTALIGN, pt.x, pt.y, \_wndBack.hwnd, null);

if (uiCmd)

{

hr = \_ToggleColumnVisibility(cast(COLUMNID)(uiCmd - IDM\_COLUMNLISTBASE));

}

}

DestroyMenu(hmnu);

}

return hr;

}

LRESULT \_OnContextMenu(UINT uiMsg, WPARAM wParam, LPARAM lParam, ref BOOL fHandled)

{

fHandled = FALSE;

HWND hwndContextMenu = cast(HWND)wParam;

// I think the listview is doing the wrong thing with WM\_CONTEXTMENU and using its own HWND even if

// the WM\_CONTEXTMENU originated in the header. Just double check the coordinates to be sure

if (hwndContextMenu == \_wndFileList.hwnd)

{

RECT rcHdr;

if (\_wndFileListHdr.GetWindowRect(&rcHdr))

{

POINT pt;

pt.x = GET\_X\_LPARAM(lParam);

pt.y = GET\_Y\_LPARAM(lParam);

if (PtInRect(&rcHdr, pt))

{

fHandled = TRUE;

\_ChooseColumns(pt);

}

}

}

return 0;

}

LRESULT \_OnDestroy(UINT uiMsg, WPARAM wParam, LPARAM lParam, ref BOOL fHandled)

{

if (\_himlToolbar)

{

\_wndToolbar.SendMessage(TB\_SETIMAGELIST, 0, cast(LPARAM)null);

ImageList\_Destroy(\_himlToolbar);

\_himlToolbar = null;

}

fHandled = TRUE;

// return CComCompositeControl<CFlatSolutionExplorer>::OnDestroy(uiMsg, wParam, lParam, fHandled);

return 0;

}

HRESULT \_OpenSolutionItem(string pszPath, int line, string scop)

{

HRESULT hr = S\_OK;

hr = OpenFileInSolutionWithScope(pszPath, line, 0, scop, true);

if(hr == S\_OK && \_closeOnReturn)

sWindowFrame.Hide();

return hr;

}

LRESULT \_OnOpenSelectedItem(WORD wNotifyCode, WORD wID, HWND hwndCtl, ref BOOL fHandled)

{

int iSel = \_wndFileList.SendMessage(LVM\_GETNEXTITEM, cast(WPARAM)-1, LVNI\_SELECTED);

if (iSel != -1)

{

\_OpenSolutionItem(iSel);

}

else

{

\_OpenSolutionItem(\_wndFileWheel.GetWindowText(), -1, "");

}

fHandled = TRUE;

return 0;

}

LRESULT \_OnFileWheelChanged(WORD wNotifyCode, WORD wID, HWND hwndCtl, ref BOOL fHandled)

{

fHandled = TRUE;

\_RefreshFileList();

return 0;

}

static struct CmdToColID

{

uint uiCmd;

COLUMNID colid;

}

static const CmdToColID[] s\_rgCmdToColIDMap =

[

// { IDR\_UNGROUPED, COLUMNID.NONE },

{ IDR\_GROUPBYKIND, COLUMNID.KIND }

];

/+

UINT \_ColumnIDtoGroupCommandID(COLUMNID colid)

{

UINT uiRet = IDR\_UNGROUPED;

BOOL fFound = FALSE;

for (int i = 0; i < s\_rgCmdToColIDMap.length && !fFound; i++)

{

if (colid == s\_rgCmdToColIDMap[i].colid)

{

uiRet = s\_rgCmdToColIDMap[i].uiCmd;

fFound = TRUE;

}

}

return uiRet;

}

+/

COLUMNID \_GroupCommandIDtoColumnID(UINT uiCmd)

{

COLUMNID colidRet = COLUMNID.NONE;

BOOL fFound = FALSE;

for (int i = 0; i < s\_rgCmdToColIDMap.length && !fFound; i++)

{

if (uiCmd == s\_rgCmdToColIDMap[i].uiCmd)

{

colidRet = s\_rgCmdToColIDMap[i].colid;

fFound = TRUE;

}

}

return colidRet;

}

HRESULT \_SetGroupColumn(COLUMNID colid)

{

\_iqp.colidGroup = colid;

\_WriteViewOptionToRegistry("GroupColumn"w, \_iqp.colidGroup);

return \_RefreshFileList();

}

int \_ListViewIndexFromColumnID(COLUMNID colid)

{

int iCol = -1;

int cCols = \_wndFileListHdr.SendMessage(HDM\_GETITEMCOUNT);

for (int i = 0; i < cCols && iCol == -1; i++)

{

HDITEM hdi;

hdi.mask = HDI\_LPARAM;

if (\_wndFileListHdr.SendMessage(HDM\_GETITEM, i, cast(LPARAM)&hdi) && hdi.lParam == colid)

{

iCol = i;

}

}

return iCol;

}

COLUMNINFO \*\_ColumnInfoFromColumnID(COLUMNID colid)

{

COLUMNINFO \*pci = null;

for (size\_t iCol = 0; iCol < \_rgColumns.length && pci is null; iCol++)

{

COLUMNINFO \*ci = &(\*\_rgColumns)[iCol];

if (ci.colid == colid)

{

pci = ci;

}

}

return pci;

}

HRESULT \_SetCompressedNameAndPath(BOOL fSet)

{

HRESULT hr = S\_OK;

if (fSet != \_fCombineColumns)

{

int iName = \_ListViewIndexFromColumnID(COLUMNID.NAME);

COLUMNID colPath = \_iqp.searchFile ? COLUMNID.PATH : COLUMNID.TYPE;

COLUMNINFO \*pciPath = \_ColumnInfoFromColumnID(colPath);

COLUMNINFO \*pciName = \_ColumnInfoFromColumnID(COLUMNID.NAME);

hr = (iName > -1 && pciPath && pciName) ? S\_OK : E\_FAIL;

if (SUCCEEDED(hr))

{

\_fCombineColumns = fSet;

\_wndFileList.SetRedraw(FALSE);

\_wndFileListHdr.SetRedraw(FALSE);

if (fSet)

{

// If the path column is currently hidden, set it to visible

if (pciPath.fVisible)

{

int iPath = \_ListViewIndexFromColumnID(colPath);

hr = \_wndFileList.SendMessage(LVM\_DELETECOLUMN, iPath) ? S\_OK : E\_FAIL;

}

else

{

pciPath.fVisible = TRUE;

}

\_wndFileList.SendMessage(LVM\_SETCOLUMNWIDTH, iName, MAKELPARAM(pciName.cx + pciPath.cx, 0));

// If the list is currently sorted by path, change it to name. Otherwise, just reset the values

// for the name column and avoid a requery

if (\_iqp.colidSort == colPath)

{

\_SetSortColumn(COLUMNID.NAME, iName);

}

else

{

LVITEM lvi;

lvi.mask = LVIF\_TEXT;

lvi.pszText = LPSTR\_TEXTCALLBACK;

lvi.iSubItem = iName;

UINT cItems = cast(UINT)\_wndFileList.SendMessage(LVM\_GETITEMCOUNT);

for (UINT i = 0; i < cItems; i++)

{

lvi.iItem = i;

\_wndFileList.SendItemMessage(LVM\_SETITEM, lvi);

}

}

}

else

{

\_wndFileList.SendMessage(LVM\_SETCOLUMNWIDTH, iName, MAKELPARAM(pciName.cx, 0));

pciPath.cx = max(pciPath.cx, 30);

hr = \_InsertListViewColumn(iName + 1, colPath, pciPath.cx);

if (SUCCEEDED(hr))

{

LVITEM lvi;

lvi.mask = LVIF\_TEXT;

lvi.pszText = LPSTR\_TEXTCALLBACK;

UINT cItems = cast(UINT)\_wndFileList.SendMessage(LVM\_GETITEMCOUNT);

for (UINT i = 0; i < cItems; i++)

{

lvi.iItem = i;

lvi.iSubItem = iName;

\_wndFileList.SendItemMessage(LVM\_SETITEM, lvi);

lvi.iSubItem = iName+1;

\_wndFileList.SendItemMessage(LVM\_SETITEM, lvi);

}

}

}

\_WriteViewOptionToRegistry("CombineColumns"w, \_fCombineColumns);

\_wndFileListHdr.SetRedraw(TRUE);

\_wndFileListHdr.InvalidateRect(null, FALSE);

\_wndFileList.SetRedraw(TRUE);

\_wndFileList.InvalidateRect(null, FALSE);

}

}

return hr;

}

LRESULT \_OnCheckBtnClicked(WORD wNotifyCode, WORD wID, HWND hwndCtl, ref BOOL fHandled)

{

TBBUTTONINFO tbbi;

tbbi.cbSize = tbbi.sizeof;

tbbi.dwMask = TBIF\_STATE;

if (\_wndToolbar.SendMessage(TB\_GETBUTTONINFO, wID, cast(LPARAM)&tbbi) != -1)

{

bool checked = !!(tbbi.fsState & TBSTATE\_CHECKED);

switch(wID)

{

case IDR\_COMBINECOLUMNS:

\_SetCompressedNameAndPath(checked);

break;

case IDR\_ALTERNATEROWCOLOR:

\_fAlternateRowColor = checked;

\_WriteViewOptionToRegistry("AlternateRowColor"w, \_fAlternateRowColor);

\_wndFileList.InvalidateRect(null, FALSE);

break;

case IDR\_CLOSEONRETURN:

\_closeOnReturn = checked;

\_WriteViewOptionToRegistry("CloseOnReturn"w, \_closeOnReturn);

break;

case IDR\_GROUPBYKIND:

\_SetGroupColumn(checked ? COLUMNID.KIND : COLUMNID.NONE);

break;

case IDR\_WHOLEWORD:

\_iqp.wholeWord = checked;

\_WriteViewOptionToRegistry("WholeWord"w, \_iqp.wholeWord);

\_RefreshFileList();

break;

case IDR\_CASESENSITIVE:

\_iqp.caseSensitive = !checked;

\_WriteViewOptionToRegistry("CaseSensitive"w, \_iqp.caseSensitive);

\_RefreshFileList();

break;

case IDR\_REGEXP:

\_iqp.useRegExp = checked;

\_WriteViewOptionToRegistry("UseRegExp"w, \_iqp.useRegExp);

\_RefreshFileList();

break;

case IDR\_SEARCHFILE:

\_ReinitViewState(checked, true);

break;

case IDR\_SEARCHSYMBOL:

\_ReinitViewState(!checked, true);

break;

default:

return 1;

}

}

fHandled = TRUE;

return 0;

}

////////////////////////////////////////////////////////////////////////

COLUMNID \_ColumnIDFromListViewIndex(int iIndex)

{

COLUMNID colid = COLUMNID.NONE;

HDITEM hdi;

hdi.mask = HDI\_LPARAM;

if (\_wndFileListHdr.SendMessage(HDM\_GETITEM, iIndex, cast(LPARAM)&hdi))

{

colid = cast(COLUMNID)hdi.lParam;

}

return colid;

}

string \_timeString(const(SysTime) time)

{

version(all)

{

DateTime dt = cast(DateTime) time;

return dt.toSimpleString();

}

else

{

char[] buffer = new char[128];

//                auto dst = daylightSavingTA(time);

//                auto offset = localTZA + dst;

auto t = time; // + offset;

auto len = sprintf(buffer.ptr, "%04d/%02d/%02d %02d:%02d:%02d",

yearFromTime(t), dateFromTime(t), monthFromTime(t) + 1,

hourFromTime(t), minFromTime(t), secFromTime(t));

assert(len < buffer.length);

buffer = buffer[0 .. len];

return assumeUnique(buffer);

}

}

////////////////////////////////////////////////////////////////////////

LRESULT \_OnFileListGetDispInfo(int idCtrl, in NMHDR \*pnmh, ref BOOL fHandled)

{

NMLVDISPINFO \*pnmlvdi = cast(NMLVDISPINFO \*)pnmh;

if (pnmlvdi.item.mask & LVIF\_TEXT)

{

LVITEM lvi;

lvi.mask = LVIF\_PARAM;

lvi.iItem = pnmlvdi.item.iItem;

if (\_wndFileList.SendItemMessage(LVM\_GETITEM, lvi))

{

pnmlvdi.item.mask |= LVIF\_DI\_SETITEM;

SolutionItem psiWeak = cast(SolutionItem)cast(void\*)lvi.lParam;

string txt;

switch (\_ColumnIDFromListViewIndex(pnmlvdi.item.iSubItem))

{

case COLUMNID.NAME:

if (\_fCombineColumns)

{

string name = psiWeak.GetName();

if(\_iqp.searchFile)

{

string path = psiWeak.GetPath();

txt = name ~ " (" ~ path ~ ")";

}

else

{

string type = psiWeak.GetType();

if(type.length)

txt = name ~ " : " ~ type;

else

txt = name;

}

}

else

{

txt = psiWeak.GetName();

}

break;

case COLUMNID.PATH:

txt = psiWeak.GetPath();

break;

case COLUMNID.SIZE:

long cb = psiWeak.GetSize();

txt = to!string(cb);

break;

case COLUMNID.MODIFIEDDATE:

const(SysTime) ft = psiWeak.GetModified();

if(ft.stdTime() != 0)

//txt = std.date.toString(ft);

txt = \_timeString(ft);

break;

case COLUMNID.LINE:

int ln = psiWeak.GetLine();

if(ln >= 0)

txt = to!string(ln);

break;

case COLUMNID.SCOPE:

txt = psiWeak.GetScope();

break;

case COLUMNID.TYPE:

txt = psiWeak.GetType();

break;

case COLUMNID.KIND:

txt = psiWeak.GetKind();

break;

default:

break;

}

wstring wtxt = toUTF16(txt) ~ '\000';

int cnt = min(wtxt.length, pnmlvdi.item.cchTextMax);

pnmlvdi.item.pszText[0..cnt] = wtxt.ptr[0..cnt];

}

}

fHandled = TRUE;

return 0;

}

void \_ReinitViewState(bool searchFile, bool refresh)

{

\_WriteViewStateToRegistry();

\_RemoveSortIcon(\_ListViewIndexFromColumnID(\_iqp.colidSort));

\_iqp.searchFile = searchFile;

\_rgColumns = \_iqp.searchFile ? &\_fileColumns : &\_symbolColumns;

\_InitializeViewState();

\_InitializeSwitches();

\_AddSortIcon(\_ListViewIndexFromColumnID(\_iqp.colidSort), \_iqp.fSortAscending);

\_InitializeFileListColumns();

\_RefreshFileList();

}

RegKey \_GetCurrentRegKey(bool write)

{

GlobalOptions opt = Package.GetGlobalOptions();

opt.getRegistryRoot();

wstring regPath = opt.regUserRoot ~ regPathToolsOptions;

if(\_iqp.searchFile)

regPath ~= "\\SearchFileWindow"w;

else

regPath ~= "\\SearchSymbolWindow"w;

return new RegKey(opt.hUserKey, regPath, write);

}

HRESULT \_InitializeViewState()

{

HRESULT hr = S\_OK;

try

{

scope RegKey keyWinOpts = \_GetCurrentRegKey(false);

if(keyWinOpts.GetDWORD("ColumnInfoVersion"w, 0) == kColumnInfoVersion)

{

void[] data = keyWinOpts.GetBinary("ColumnInfo"w);

if(data !is null)

\*\_rgColumns = cast(COLUMNINFO[])data;

}

\_iqp.colidSort = cast(COLUMNID) keyWinOpts.GetDWORD("SortColumn"w, \_iqp.colidSort);

\_iqp.colidGroup = cast(COLUMNID) keyWinOpts.GetDWORD("GroupColumn"w, \_iqp.colidGroup);

\_iqp.fSortAscending = keyWinOpts.GetDWORD("SortAscending"w, \_iqp.fSortAscending) != 0;

\_iqp.wholeWord = keyWinOpts.GetDWORD("WholeWord"w, \_iqp.wholeWord) != 0;

\_iqp.caseSensitive = keyWinOpts.GetDWORD("CaseSensitive"w, \_iqp.caseSensitive) != 0;

\_iqp.useRegExp = keyWinOpts.GetDWORD("UseRegExp"w, \_iqp.useRegExp) != 0;

\_fCombineColumns = keyWinOpts.GetDWORD("CombineColumns"w, \_fCombineColumns) != 0;

\_fAlternateRowColor = keyWinOpts.GetDWORD("AlternateRowColor"w, \_fAlternateRowColor) != 0;

\_closeOnReturn = keyWinOpts.GetDWORD("closeOnReturn"w, \_closeOnReturn) != 0;

}

catch(Exception e)

{

// ok to fail, defaults still work

}

return hr;

}

HRESULT \_WriteViewStateToRegistry()

{

\_WriteColumnInfoToRegistry();

HRESULT hr = S\_OK;

try

{

scope RegKey keyWinOpts = \_GetCurrentRegKey(true);

keyWinOpts.Set("SortColumn"w, \_iqp.colidSort);

keyWinOpts.Set("GroupColumn"w, \_iqp.colidGroup);

keyWinOpts.Set("SortAscending"w, \_iqp.fSortAscending);

keyWinOpts.Set("WholeWord"w, \_iqp.wholeWord);

keyWinOpts.Set("CaseSensitive"w, \_iqp.caseSensitive);

keyWinOpts.Set("UseRegExp"w, \_iqp.useRegExp);

keyWinOpts.Set("CombineColumns"w, \_fCombineColumns);

keyWinOpts.Set("AlternateRowColor"w, \_fAlternateRowColor);

keyWinOpts.Set("closeOnReturn"w, \_closeOnReturn);

}

catch(Exception e)

{

hr = E\_FAIL;

}

return hr;

}

HRESULT \_WriteColumnInfoToRegistry()

{

HRESULT hr = S\_OK;

for(int i = 0; i < \_rgColumns.length; i++)

(\*\_rgColumns)[i].cx = \_wndFileList.SendMessage(LVM\_GETCOLUMNWIDTH, \_ListViewIndexFromColumnID((\*\_rgColumns)[i].colid));

try

{

scope RegKey keyWinOpts = \_GetCurrentRegKey(true);

keyWinOpts.Set("ColumnInfoVersion"w, kColumnInfoVersion);

keyWinOpts.Set("ColumnInfo"w, \*\_rgColumns);

}

catch(Exception e)

{

hr = E\_FAIL;

}

return hr;

}

HRESULT \_WriteViewOptionToRegistry(wstring name, DWORD dw)

{

HRESULT hr = S\_OK;

try

{

scope RegKey keyWinOpts = \_GetCurrentRegKey(true);

keyWinOpts.Set(toUTF16(name), dw);

}

catch(Exception e)

{

hr = E\_FAIL;

}

return hr;

}

HRESULT \_WriteSortInfoToRegistry()

{

HRESULT hr = S\_OK;

try

{

scope RegKey keyWinOpts = \_GetCurrentRegKey(true);

keyWinOpts.Set("SortColumn"w, \_iqp.colidSort);

keyWinOpts.Set("SortAscending"w, \_iqp.fSortAscending);

}

catch(Exception e)

{

hr = E\_FAIL;

}

return hr;

}

HRESULT \_SetSortColumn(COLUMNID colid, int iIndex)

{

HRESULT hr = S\_OK;

bool fSortAscending = true;

if (colid == \_iqp.colidSort)

{

fSortAscending = !\_iqp.fSortAscending;

}

else

{

int iIndexCur = \_ListViewIndexFromColumnID(\_iqp.colidSort);

if (iIndexCur != -1) // Current sort column may have been removed from the list view

{

hr = \_RemoveSortIcon(iIndexCur);

}

}

if (SUCCEEDED(hr))

{

hr = \_AddSortIcon(iIndex, fSortAscending);

if (SUCCEEDED(hr))

{

\_iqp.colidSort = colid;

\_iqp.fSortAscending = fSortAscending;

\_WriteSortInfoToRegistry();

hr = \_RefreshFileList();

}

}

return hr;

}

LRESULT \_OnFileListColumnClick(int idCtrl, ref NMHDR \*pnmh, ref BOOL fHandled)

{

NMLISTVIEW \*pnmlv = cast(NMLISTVIEW \*)pnmh;

\_SetSortColumn(\_ColumnIDFromListViewIndex(pnmlv.iSubItem), pnmlv.iSubItem);

fHandled = TRUE;

return 0;

}

LRESULT \_OnFileListDeleteItem(int idCtrl, ref NMHDR \*pnmh, ref BOOL fHandled)

{

NMLISTVIEW \*pnmlv = cast(NMLISTVIEW \*)pnmh;

SolutionItem psi = cast(SolutionItem)cast(void\*)pnmlv.lParam;

// psi.Release();

fHandled = TRUE;

return 0;

}

HRESULT \_OpenSolutionItem(int iIndex)

{

LVITEM lvi;

lvi.mask = LVIF\_PARAM;

lvi.iItem = iIndex;

HRESULT hr = \_wndFileList.SendItemMessage(LVM\_GETITEM, lvi) ? S\_OK : E\_FAIL;

if (SUCCEEDED(hr))

{

SolutionItem psiWeak = cast(SolutionItem)cast(void\*)lvi.lParam;

string fname = psiWeak.GetFullPath();

version(none)

{

string scop = !\_iqp.searchFile ? psiWeak.GetScope() : null;

hr = \_OpenSolutionItem(fname, psiWeak.GetLine(), scop);

}

else

{

hr = \_OpenSolutionItem(fname, psiWeak.GetLine(), "");

if(hr != S\_OK && !\_iqp.searchFile && !isAbsolute(fname))

{

// guess import path from filename (e.g. "src\core\mem.d") and

// scope (e.g. "core.mem.gc.Proxy") to try opening

// the file ("core\mem.d")

string inScope = toLower(psiWeak.GetScope());

string path = normalizeDir(dirName(toLower(psiWeak.GetPath())));

inScope = replace(inScope, ".", "\\");

int i;

for(i = 1; i < path.length; i++)

if(startsWith(inScope, path[i .. $]))

break;

if(i < path.length)

{

fname = fname[i .. $];

hr = \_OpenSolutionItem(fname, psiWeak.GetLine(), "");

}

}

}

}

return hr;

}

LRESULT \_OnFileListDblClick(int idCtrl, ref NMHDR \*pnmh, ref BOOL fHandled)

{

NMITEMACTIVATE \*pnmitem = cast(NMITEMACTIVATE\*) pnmh;

if (FAILED(\_OpenSolutionItem(pnmitem.iItem)))

{

MessageBeep(MB\_ICONHAND);

}

fHandled = TRUE;

return 0;

}

void \_SetAlternateRowColor()

{

COLORREF cr = GetSysColor(COLOR\_HIGHLIGHT);

BYTE r = GetRValue(cr);

BYTE g = GetGValue(cr);

BYTE b = GetBValue(cr);

BYTE rNew = 236;

BYTE gNew = 236;

BYTE bNew = 236;

if (r > g && r > b)

{

rNew = 244;

}

else if (g > r && g > b)

{

gNew = 244;

}

else

{

bNew = 244;

}

\_crAlternate = RGB(rNew, gNew, bNew);

}

LRESULT \_OnFileListCustomDraw(int idCtrl, ref NMHDR \*pnmh, ref BOOL fHandled)

{

LRESULT lRet = CDRF\_DODEFAULT;

NMLVCUSTOMDRAW \*pnmlvcd = cast(NMLVCUSTOMDRAW \*)pnmh;

switch (pnmlvcd.nmcd.dwDrawStage)

{

case CDDS\_PREPAINT:

\_SetAlternateRowColor();

lRet = CDRF\_NOTIFYITEMDRAW;

break;

case CDDS\_ITEMPREPAINT:

{

// Override the colors so that regardless of the focus state, the control appears focused.

// We can't rely on the pnmlvcd.nmcd.uItemState for this because there is a known bug

// with listviews that have the LVS\_EX\_SHOWSELALWAYS style where this bit is set for

// every item

LVITEM lvi;

lvi.mask = LVIF\_STATE;

lvi.iItem = cast(int)pnmlvcd.nmcd.dwItemSpec;

lvi.stateMask = LVIS\_SELECTED;

if (\_wndFileList.SendItemMessage(LVM\_GETITEM, lvi) && (lvi.state & LVIS\_SELECTED))

{

pnmlvcd.clrText = GetSysColor(COLOR\_HIGHLIGHTTEXT);

pnmlvcd.clrTextBk = GetSysColor(COLOR\_HIGHLIGHT);

pnmlvcd.nmcd.uItemState &= ~CDIS\_SELECTED;

lRet = CDRF\_NEWFONT;

}

else

{

if (\_fAlternateRowColor && !(pnmlvcd.nmcd.dwItemSpec % 2))

{

// TODO: Eventually, it might be nice to build a color based on COLOR\_HIGHLIGHT.

pnmlvcd.clrTextBk = \_crAlternate;

pnmlvcd.nmcd.uItemState &= ~CDIS\_SELECTED;

lRet = CDRF\_NEWFONT;

}

}

break;

}

default:

break;

}

fHandled = TRUE;

return lRet;

}

LRESULT \_OnFileListHdrItemChanged(int idCtrl, ref NMHDR \*pnmh, ref BOOL fHandled)

{

NMHEADER \*pnmhdr = cast(NMHEADER \*)pnmh;

if (pnmhdr.pitem.mask & HDI\_WIDTH)

{

COLUMNID colid = \_ColumnIDFromListViewIndex(pnmhdr.iItem);

if (colid == COLUMNID.NAME && \_fCombineColumns)

{

// Get the size delta and distrubute it between the name and path columns

COLUMNID colPath = \_iqp.searchFile ? COLUMNID.PATH : COLUMNID.TYPE;

COLUMNINFO \*pciName = \_ColumnInfoFromColumnID(COLUMNID.NAME);

COLUMNINFO \*pciPath = \_ColumnInfoFromColumnID(colPath);

int cxTotal = pciName.cx + pciPath.cx;

int cxDelta = pnmhdr.pitem.cxy - cxTotal;

int iPercentChange = MulDiv(100, cxDelta, cxTotal);

int cxNameDelta = MulDiv(abs(cxDelta), iPercentChange, 100);

int cxPathDelta = cxDelta - cxNameDelta;

pciName.cx += cxNameDelta;

pciPath.cx += cxPathDelta;

}

else

{

COLUMNINFO \*pci = \_ColumnInfoFromColumnID(colid);

pci.cx = pnmhdr.pitem.cxy;

}

\_WriteColumnInfoToRegistry();

}

fHandled = TRUE;

return 0;

}

LRESULT \_OnToolbarGetInfoTip(int idCtrl, ref NMHDR \*pnmh, ref BOOL fHandled)

{

NMTBGETINFOTIP \*pnmtbgit = cast(NMTBGETINFOTIP \*)pnmh;

string tip;

switch(pnmtbgit.iItem)

{

case IDR\_COMBINECOLUMNS:

if(\_iqp.searchFile)

tip = "Toggle single/double column display of name and path";

else

tip = "Toggle single/double column display of name and type";

break;

case IDR\_ALTERNATEROWCOLOR:

tip = "Toggle alternating row background color";

break;

case IDR\_GROUPBYKIND:

tip = "Grouped display by kind";

break;

case IDR\_CLOSEONRETURN:

tip = "Close search window when item selected or focus lost";

break;

case IDR\_WHOLEWORD:

tip = "Match whole word only";

break;

case IDR\_CASESENSITIVE:

tip = "Match case insensitive";

break;

case IDR\_REGEXP:

tip = "Match by regular expression";

break;

case IDR\_SEARCHFILE:

tip = "Search for file in solution";

break;

case IDR\_SEARCHSYMBOL:

tip = "Search for symbol in solution";

break;

default:

break;

}

wstring wtip = toUTF16(tip) ~ '\000';

int cnt = min(wtip.length, pnmtbgit.cchTextMax);

pnmtbgit.pszText[0..cnt] = wtip.ptr[0..cnt];

fHandled = TRUE;

return 0;

}

}

////////////////////////////////////////////////////////////////////////

class SolutionItem //: IUnknown

{

static const GUID iid = uuid("6EB1B172-33C2-418a-8B67-F428FD456B46");

this(string path, string relpath)

{

int idx = lastIndexOf(path, '\\');

if(idx < 0)

def.name = path;

else

def.name = path[idx + 1 .. $];

def.filename = path;

def.line = -1;

def.kind = "file";

if(exists(path))

\_modifiedDate = timeLastModified(path);

def.inScope = relpath;

}

this(Definition d)

{

def = d;

if(def.kind == "module")

def.line = -1;

}

int GetIconIndex() const { return 0; }

string GetName() const

{

return def.name;

}

string GetFullPath() const

{

return def.filename;

}

string GetPath() const

{

if(def.kind != "file")

return def.filename;

if(def.inScope.length)

return def.inScope;

int idx = lastIndexOf(def.filename, '\\');

if(idx < 0)

return "";

return def.filename[0 .. idx];

}

int GetLine() const { return def.line; }

string GetScope() const { return def.inScope; }

string GetType() const { return def.type; }

string GetKind() const { return def.kind; }

long GetSize() const { return 0; }

const(SysTime) GetModified() const { return \_modifiedDate; }

//HRESULT GetItem(in IID\* riid, void \*\*ppv);

Definition def;

SysTime \_modifiedDate;

}

class SolutionItemGroup //: IUnknown

{

static const GUID iid = uuid("FCF2F784-0C4E-4c2c-A0CE-E44E3B20D8E2");

this(string name)

{

mName = name;

mArray = new ItemArray;

}

void add(SolutionItem item)

{

mArray.add(item);

}

string GetName() const { return mName; }

const(ItemArray) GetItems() const { return mArray; }

ItemArray mArray;

string mName;

}

class SolutionItemIndex //: IUnknown

{

static const GUID iid = uuid("DA2FC9FF-57D4-42bd-9E26-518A42668DEE");

HRESULT Search(string pszSearch, INDEXQUERYPARAMS \*piqp, ItemArray \*ppv)

{

string[] args = tokenizeArgs(pszSearch);

auto arr = new ItemArray;

SearchData sd;

sd.wholeWord = piqp.wholeWord;

sd.caseSensitive = piqp.caseSensitive;

sd.useRegExp = piqp.useRegExp;

if(!sd.init(args))

return E\_FAIL;

if (piqp.searchFile)

{

string solutionpath = GetSolutionFilename();

string solutiondir = normalizeDir(dirName(solutionpath));

searchSolutionItem(delegate bool(string s)

{

string f = s;

if(s.startsWith(solutiondir)) // case-insensitive?

f = s[solutiondir.length .. $];

//makeRelative(s, solutiondir);

if(!sd.matchNames(f, "", "", ""))

return false;

if(f == s)

f = "";

if(piqp.colidGroup == COLUMNID.KIND)

{

string ext = extension(s);

if (!arr.getItemByGroupAndPath(ext, s))

arr.addByGroup(ext, new SolutionItem(s, f));

}

else

{

if (!arr.getItemByPath(s))

arr.add(new SolutionItem(s, f));

}

return false;

});

}

else

{

Definition[] defs = Package.GetLibInfos().findDefinition(sd);

foreach(ref def; defs)

{

if(piqp.colidGroup == COLUMNID.KIND)

arr.addByGroup(def.kind, new SolutionItem(def));

else

arr.add(new SolutionItem(def));

}

}

arr.sort(piqp.colidSort, piqp.fSortAscending);

\*ppv = arr;

return S\_OK;

}

}

class ItemArray //: IUnknown

{

static const GUID iid = uuid("5A97C4DF-DE3A-4bb6-B621-2F9550BFE7C0");

SolutionItem[string] mItemsByPath;

SolutionItem[] mItems;

SolutionItemGroup[] mGroups;

this()

{

}

const(SolutionItem) getItemByPath(string path) const

{

if (auto it = path in mItemsByPath)

return \*it;

return null;

}

void add(SolutionItem item)

{

mItems ~= item;

mItemsByPath[item.GetFullPath()] = item;

}

const(SolutionItem) getItemByGroupAndPath(string grp, string path)

{

for(int i = 0; i < mGroups.length; i++)

if(mGroups[i].GetName() == grp)

return mGroups[i].GetItems().getItemByPath(path);

return null;

}

void addByGroup(string grp, SolutionItem item)

{

for(int i = 0; i < mGroups.length; i++)

if(mGroups[i].GetName() == grp)

return mGroups[i].add(item);

auto group = new SolutionItemGroup(grp);

group.add(item);

mGroups ~= group;

}

int GetCount() const { return max(mItems.length, mGroups.length); }

SolutionItemGroup GetGroup(uint idx) const

{

if(idx >= mGroups.length)

return null;

return cast(SolutionItemGroup)mGroups[idx];

}

SolutionItem GetItem(uint idx) const

{

if(idx >= mItems.length)

return null;

return cast(SolutionItem)mItems[idx];

}

//HRESULT GetItem(I)(uint idx, I\*ptr) const { return E\_FAIL; }

void sort(COLUMNID id, bool ascending)

{

switch(id)

{

case COLUMNID.NAME:

if(ascending)

std.algorithm.sort!("a.GetName() < b.GetName()")(mItems);

else

std.algorithm.sort!("a.GetName() > b.GetName()")(mItems);

break;

case COLUMNID.LINE:

if(ascending)

std.algorithm.sort!("a.GetLine() < b.GetLine()")(mItems);

else

std.algorithm.sort!("a.GetLine() > b.GetLine()")(mItems);

break;

case COLUMNID.TYPE:

if(ascending)

std.algorithm.sort!("a.GetType() < b.GetType()")(mItems);

else

std.algorithm.sort!("a.GetType() > b.GetType()")(mItems);

break;

case COLUMNID.PATH:

if(ascending)

std.algorithm.sort!("a.GetPath() < b.GetPath()")(mItems);

else

std.algorithm.sort!("a.GetPath() > b.GetPath()")(mItems);

break;

case COLUMNID.SCOPE:

if(ascending)

std.algorithm.sort!("a.GetScope() < b.GetScope()")(mItems);

else

std.algorithm.sort!("a.GetScope() > b.GetScope()")(mItems);

break;

case COLUMNID.MODIFIEDDATE:

if(ascending)

std.algorithm.sort!("a.GetModified() < b.GetModified()")(mItems);

else

std.algorithm.sort!("a.GetModified() > b.GetModified()")(mItems);

break;

default:

break;

}

foreach(grp; mGroups)

grp.mArray.sort(id, ascending);

}

}

////////////////////////////////////////////////////////////////////////

bool searchHierarchy(IVsHierarchy pHierarchy, VSITEMID item, bool delegate (string) dg)

{

VARIANT var;

if((pHierarchy.GetProperty(item, VSHPROPID\_Container, &var) == S\_OK &&

((var.vt == VT\_BOOL && var.boolVal) || (var.vt == VT\_I4 && var.lVal))) ||

(pHierarchy.GetProperty(item, VSHPROPID\_Expandable, &var) == S\_OK &&

((var.vt == VT\_BOOL && var.boolVal) || (var.vt == VT\_I4 && var.lVal))))

{

if(pHierarchy.GetProperty(item, VSHPROPID\_FirstChild, &var) == S\_OK &&

(var.vt == VT\_INT\_PTR || var.vt == VT\_I4 || var.vt == VT\_INT))

{

VSITEMID chid = var.lVal;

while(chid != VSITEMID\_NIL)

{

if(searchHierarchy(pHierarchy, chid, dg))

return true;

if(pHierarchy.GetProperty(chid, VSHPROPID\_NextSibling, &var) != S\_OK ||

(var.vt != VT\_INT\_PTR && var.vt != VT\_I4 && var.vt != VT\_INT))

break;

chid = var.lVal;

}

}

else

{

IVsHierarchy nestedHierarchy;

VSITEMID itemidNested;

if(pHierarchy.GetNestedHierarchy(item, &IVsHierarchy.iid, cast(void \*\*)&nestedHierarchy, &itemidNested) == S\_OK)

{

if(searchHierarchy(nestedHierarchy, itemidNested, dg))

return true;

}

}

}

else if(IVsProject prj = qi\_cast!IVsProject(pHierarchy))

{

scope(exit) release(prj);

BSTR bstrMkDocument;

if(prj.GetMkDocument(item, &bstrMkDocument) == S\_OK)

{

string docname = detachBSTR(bstrMkDocument);

if(dg(docname))

return true;

}

}

return false;

}

bool searchSolutionItem(bool delegate (string) dg)

{

if(auto srpSolution = queryService!(IVsSolution))

{

scope(exit) release(srpSolution);

IEnumHierarchies pEnum;

if(srpSolution.GetProjectEnum(EPF\_LOADEDINSOLUTION, &GUID\_NULL, &pEnum) == S\_OK)

{

scope(exit) release(pEnum);

IVsHierarchy pHierarchy;

while(pEnum.Next(1, &pHierarchy, null) == S\_OK)

{

scope(exit) release(pHierarchy);

VSITEMID itemid = VSITEMID\_ROOT;

if(searchHierarchy(pHierarchy, VSITEMID\_ROOT, dg))

return true;

}

}

}

return false;

}

//------------------------------------------------------------------------------

// CSolutionItemTypeCache

//------------------------------------------------------------------------------

struct TYPECACHEINFO

{

string szFriendlyName;

int iIconIndex;

}

class CSolutionItemTypeCache

{

this()

{

\_himl = ImageList\_Create(GetSystemMetrics(SM\_CXSMICON), GetSystemMetrics(SM\_CYSMICON), ILC\_COLOR32 | ILC\_MASK, 16, 8);

}

~this()

{

if (\_himl)

ImageList\_Destroy(\_himl);

}

const(TYPECACHEINFO) \*GetTypeInfo(string pszCanonicalType)

{

if(TYPECACHEINFO\* ti = pszCanonicalType in \_mapTypes)

return ti;

SHFILEINFO shfi;

if(SHGetFileInfoW(\_toUTF16z(pszCanonicalType), FILE\_ATTRIBUTE\_NORMAL, &shfi, shfi.sizeof,

SHGFI\_ICON | SHGFI\_SMALLICON | SHGFI\_SHELLICONSIZE | SHGFI\_TYPENAME | SHGFI\_USEFILEATTRIBUTES))

{

TYPECACHEINFO tci;

tci.iIconIndex = ImageList\_ReplaceIcon(\_himl, -1, shfi.hIcon);

if(tci.iIconIndex != -1)

{

tci.szFriendlyName = to\_string(shfi.szTypeName.ptr);

\_mapTypes[pszCanonicalType] = tci;

}

DestroyIcon(shfi.hIcon);

}

return pszCanonicalType in \_mapTypes;

}

HIMAGELIST GetIconImageList() { return \_himl; }

private:

TYPECACHEINFO[string] \_mapTypes;

HIMAGELIST \_himl;

};

// This file is part of Visual D

//

// Visual D integrates the D programming language into Visual Studio

// Copyright (c) 2010 by Rainer Schuetze, All Rights Reserved

//

// Distributed under the Boost Software License, Version 1.0.

// See accompanying file LICENSE\_1\_0.txt or copy at <http://www.boost.org/LICENSE_1_0.txt>

module visuald.simpleparser;

import std.exception;

import std.string;

import std.conv;

import std.array;

import vdc.lexer;

version(MAIN)

{

import std.stdio;

debug = log;

}

else // !version(MAIN)

{

import sdk.vsi.sdk\_shared;

import visuald.logutil;

//debug = log;

alias logCall writeln;

}

// very simple parser, just checking curly braces and statement/declaration boundaries

// we are mainly interested in finding the matching else to if, version and debug statements

/\* Grammar:

Module:

Statements

Statements:

Statement

Statement Statements

Statement:

IfStatement

VersionStatement

DebugStatement

ScopedStatement

;

OtherToken Statement

ScopedStatement:

{ }

{ Statements }

IfStatement:

if(Expression) Statement

if(Expression) Statement else Statement

VersionStatement:

version(Expression) Statement

version(Expression) Statement else Statement

DebugStatement:

debug(Expression) Statement

debug(Expression) Statement else Statement

Expression:

BracedExpression

BracedExpression Expression

NonBraceToken Expression

BracedExpression:

( Expression )

[ Expression ]

ScopedStatement

OtherToken: anything but '{', ';', if, version, debug (might also exclude ')', ']', '}')

NonBraceToken: anything but '{', '(', '[' (might also exclude ')', ']', '}')

\*/

//////////////////////////////////////////////////////////////////////////////

\_\_gshared Lexer spLex;

struct ParserSpan

{

int iStartIndex; // starting character index within the line (must be <= length of line)

int iStartLine; // starting line

int iEndIndex; // ending character index within the line (must be <= length of line)

int iEndLine; // ending line

}

struct ParserToken(S)

{

S text;

int type;

int id;

ParserSpan span;

};

string logString(ref ParserSpan span)

{

if(span.iStartLine == 0 && span.iEndLine == 0)

return "[" ~ to!string(span.iStartIndex) ~ "," ~ to!string(span.iEndIndex) ~ "]";

return "[" ~ to!string(span.iStartLine) ~ ":" ~ to!string(span.iStartIndex)

~ "," ~ to!string(span.iEndLine) ~ ":" ~ to!string(span.iEndIndex) ~ "]";

}

// returns < 0 if adr1 < adr2

int compareTextAddress(int line1, int index1, int line2, int index2)

{

int difflines = line1 - line2;

if(difflines != 0)

return difflines;

return index1 - index2;

}

int compareStartAddress(ref const(ParserSpan) span, int line, int index)

{

return compareTextAddress(span.iStartLine, span.iStartIndex, line, index);

}

int compareEndAddress(ref const(ParserSpan) span, int line, int index)

{

return compareTextAddress(span.iEndLine, span.iEndIndex, line, index);

}

bool spanContains(ref const(ParserSpan) span, int line, int index)

{

return compareStartAddress(span, line, index) <= 0 && compareEndAddress(span, line, index) > 0;

}

bool spanEmpty(ref const(ParserSpan) span)

{

return span.iStartLine == span.iEndLine && span.iStartIndex == span.iEndIndex;

}

//////////////////////////////////////////////////////////////

class LocationBase(S)

{

alias ParserBase!S Parser;

alias LocationBase!S Location;

Location parent;

Location[] children;

ParserSpan span;

this(Location \_parent, ParserSpan \_span)

{

span.iStartLine = span.iEndLine = \_span.iStartLine;

span.iStartIndex = span.iEndIndex = \_span.iStartIndex;

parent = \_parent;

}

mixin template ForwardConstructor()

{

this(Location \_parent, ParserSpan \_span)

{

super(\_parent, \_span);

}

}

void extendSpan(ref ParserSpan \_span)

{

span.iEndLine = \_span.iEndLine;

span.iEndIndex = \_span.iEndIndex;

}

void limitSpan(ref ParserSpan \_span)

{

span.iEndLine = \_span.iEndLine;

span.iEndIndex = \_span.iEndIndex;

}

void clearSpan()

{

span.iEndLine = span.iStartLine;

span.iEndIndex = span.iStartIndex;

}

// return true if token consumed

abstract bool shift(Parser parser, ref ParserToken!S tok);

// return true if reduce should not be called on parent

bool reduce(Parser parser, Location loc)

{

extendSpan(loc.span);

return true;

}

bool isStatement()

{

return true;

}

}

class Module(S) : LocationBase!S

{

this()

{

ParserSpan \_span;

super(null, \_span);

}

override bool shift(Parser parser, ref ParserToken!S tok)

{

Location loc;

switch(tok.id)

{

case TOK\_rparen:

case TOK\_rbracket:

case TOK\_rcurly: // mismatched brace - do not create statement, it will reduce on them, just eat away

return true;

case TOK\_if:

loc = new IfStatement!S(this, tok.span);

break;

case TOK\_version:

loc = new VersionStatement!S(this, tok.span);

break;

case TOK\_debug:

loc = new DebugStatement!S(this, tok.span);

break;

case TOK\_unittest:

loc = new UnittestStatement!S(this, tok.span);

break;

default:

Statement!S stmt = new Statement!S(this, tok.span);

parser.push(stmt);

return false;

}

parser.push(loc);

return false;

}

}

// children are braced sub expressions, the last child might be a trailing statement as in

// scope(exit) { foo() }

class Statement(S) : LocationBase!S

{

mixin ForwardConstructor;

override bool shift(Parser parser, ref ParserToken!S tok)

{

Location loc;

switch(tok.id)

{

case TOK\_if:

loc = new IfStatement!S(this, tok.span);

break;

case TOK\_version:

loc = new VersionStatement!S(this, tok.span);

break;

case TOK\_debug:

loc = new DebugStatement!S(this, tok.span);

break;

case TOK\_unittest:

loc = new UnittestStatement!S(this, tok.span);

break;

case TOK\_lcurly:

loc = new CurlyBracedStatement!S(this, tok.span);

break;

case TOK\_lbracket:

loc = new SquareBracedExpression!S(this, tok.span);

break;

case TOK\_lparen:

loc = new RoundBracedExpression!S(this, tok.span);

break;

case TOK\_rparen:

case TOK\_rbracket:

case TOK\_rcurly: // mismatched brace - bail out

parser.reduce();

return false;

case TOK\_semicolon:

extendSpan(tok.span);

parser.reduce();

return true;

default:

extendSpan(tok.span);

return true;

}

parser.push(loc);

return false;

}

override bool reduce(Parser parser, Location loc)

{

super.reduce(parser, loc);

return !loc.isStatement(); // statement always trails

}

}

class BracedStatement(S, int openid, int closeid) : LocationBase!S

{

mixin ForwardConstructor;

override bool shift(Parser parser, ref ParserToken!S tok)

{

if(spanEmpty(span))

{

extendSpan(tok.span);

assert(tok.id == openid);

return true;

}

extendSpan(tok.span);

if(tok.id == closeid)

{

parser.reduce();

return true;

}

if(tok.id == TOK\_rcurly || tok.id == TOK\_rbracket || tok.id == TOK\_rparen)

{

// mismatched brace - bail out

parser.reduce();

return false;

}

Statement!S stmt = new Statement!S(this, tok.span);

parser.push(stmt);

return false;

}

}

class CurlyBracedStatement(S) : BracedStatement!(S, TOK\_lcurly, TOK\_rcurly)

{

mixin ForwardConstructor;

}

class UnittestStatement(S) : CurlyBracedStatement!S

{

mixin ForwardConstructor;

override bool shift(Parser parser, ref ParserToken!S tok)

{

if(spanEmpty(span))

{

extendSpan(tok.span);

assert(tok.id == TOK\_unittest);

return true;

}

// if we have only parsed the "unittest", skip the following '{'

bool testCurly = span.iStartLine == span.iEndLine && span.iEndIndex == span.iStartIndex + 8;

if(testCurly)

{

if(tok.id == TOK\_lcurly)

{

extendSpan(tok.span);

return true;

}

parser.reduce(); // bail out

return false;

}

return super.shift(parser, tok);

}

override bool reduce(Parser parser, Location loc)

{

super.reduce(parser, loc);

return false; // always continue reduce after unittest statement

}

}

// do not eat closing } when reducing

class OpenCurlyBracedStatement(S) : CurlyBracedStatement!(S)

{

mixin ForwardConstructor;

override bool shift(Parser parser, ref ParserToken!S tok)

{

if(tok.id == TOK\_rcurly)

{

parser.reduce();

return false;

}

extendSpan(tok.span);

return super.shift(parser, tok);

}

}

class SquareBracedExpression(S) : BracedStatement!(S, TOK\_lbracket, TOK\_rbracket)

{

mixin ForwardConstructor;

override bool isStatement()

{

return false;

}

}

class RoundBracedExpression(S) : BracedStatement!(S, TOK\_lparen, TOK\_rparen)

{

mixin ForwardConstructor;

override bool isStatement()

{

return false;

}

}

class IfDebugVersionStatement(S, string keyword) : LocationBase!S

{

mixin ForwardConstructor;

override bool shift(Parser parser, ref ParserToken!S tok)

{

if(spanEmpty(span))

{

assert(tok.text == keyword);

extendSpan(tok.span);

return true;

}

if(children.length == 0)

{

if(tok.id != TOK\_lparen)

{

if(keyword == "debug" && tok.id != TOK\_assign)

{

ParserSpan sp = ParserSpan(tok.span.iStartIndex, tok.span.iStartLine,

tok.span.iStartIndex, tok.span.iStartLine);

children ~= new RoundBracedExpression!S(this, sp);

goto then\_statement;

}

// bail out, it's a standard statement

parser.replace(new Statement!S(parent, span));

return false;

}

extendSpan(tok.span);

Location loc = new RoundBracedExpression!S(this, tok.span);

parser.push(loc);

return false;

}

if(children.length == 1)

{

then\_statement:

extendSpan(tok.span);

if(tok.id == TOK\_colon)

{

// version(X): is terminated by EOF or }, so

// treat version(X): stmts

// as version(X) { stmts }

auto stmt = new OpenCurlyBracedStatement!S(this, tok.span);

stmt.span.iStartLine = stmt.span.iEndLine = tok.span.iEndLine;

stmt.span.iStartIndex = stmt.span.iEndIndex = tok.span.iEndIndex;

parser.push(stmt);

return true;

}

Statement!S stmt = new Statement!S(this, tok.span);

parser.push(stmt);

return false;

}

if(children.length == 2)

{

if(tok.id != TOK\_else)

{

parser.reduce();

return false;

}

extendSpan(tok.span);

Statement!S stmt = new Statement!S(this, tok.span);

parser.push(stmt);

return true;

}

parser.reduce();

return false;

}

override bool reduce(Parser parser, Location loc)

{

super.reduce(parser, loc);

return (children.length <= 2); // always continue reduce after else statement

}

}

class IfStatement(S) : IfDebugVersionStatement!(S, "if")

{

mixin ForwardConstructor;

}

class VersionStatement(S) : IfDebugVersionStatement!(S, "version")

{

mixin ForwardConstructor;

}

class DebugStatement(S) : IfDebugVersionStatement!(S, "debug")

{

mixin ForwardConstructor;

}

//////////////////////////////////////////////////////////////

class ParserBase(S = string)

{

alias ParserBase!S Parser;

alias LocationBase!S Location;

Location[] stack;

this()

{

}

void shift(ref ParserToken!S tok)

{

if(stack.length == 0)

stack ~= new Module!S;

debug(log) writeln(replicate(" ", stack.length), "shift ", tok.text, " ", logString(tok.span));

while(!stack[$-1].shift(this, tok)) {}

}

void reduce()

{

Location loc;

do

{

loc = pop();

enforce(loc.parent, "parser location has no parent");

} while(!loc.parent.reduce(this, loc));

}

void push(Location loc)

{

debug(log) writeln(replicate(" ", stack.length), "push ", loc);

assert(stack.length > 0);

assert(loc.parent == stack[$-1]);

stack[$-1].children ~= loc;

stack ~= loc;

}

Location pop()

{

enforce(stack.length, "parser stack empty");

Location loc = stack[$-1];

stack = stack[0..$-1];

debug(log) writeln(replicate(" ", stack.length), "pop ", loc, " ", logString(loc.span));

return loc;

}

void replace(Location loc)

{

debug(log) writeln(replicate(" ", stack.length), "replace ", loc);

Location prev = pop();

assert(stack.length > 0);

assert(stack[$-1].children.length > 0);

assert(stack[$-1].children[$-1] == prev);

stack[$-1].children = stack[$-1].children[0..$-1];

push(loc);

}

// throw away anything that is later than the given address

bool prune(ref int line, ref int index)

{

debug(log) writeln("prune at [", line, ":", index, "]");

static void pruneLater(int line, int index, ref Location[] locations)

{

while(locations.length > 0)

{

Location loc = locations[$-1];

if(compareStartAddress(loc.span, line, index) < 0)

break;

debug(log) writeln(" stack pruning ", loc, " at ", logString(loc.span));

locations = locations[0..$-1];

}

}

// remove stack entries that start later than the given address

pruneLater(line, index, stack);

while(stack.length > 0)

{

Location loc = stack[$-1];

assert(compareStartAddress(loc.span, line, index) < 0);

// remove children that start later than the given address

pruneLater(line, index, loc.children);

if(loc.children.length <= 0)

break;

// move child containing the the given address back on the stack

Location child = loc.children[$-1];

assert(compareStartAddress(child.span, line, index) < 0);

if(compareEndAddress(child.span, line, index) < 0)

break;

debug(log) writeln(" child pruning ", child, " at ", logString(child.span));

//                        loc.children = loc.children[0..$-1];

stack ~= child;

}

// fix span of stack entries

foreach(loc; stack)

{

if(loc.children.length)

loc.limitSpan(loc.children[$-1].span);

else

loc.clearSpan();

}

if(stack.length > 0)

{

line = stack[$-1].span.iEndLine;

index = stack[$-1].span.iEndIndex;

}

debug(log) writeln("prune returns [", line, ":", index, "]");

return true;

}

void fixExtend()

{

// fix span of stack entries

foreach\_reverse(loc; stack)

{

if(loc.children.length)

loc.extendSpan(loc.children[$-1].span);

}

}

Location findLocation(int line, int index, bool lastLocOpen)

{

static Location findLocation(Location[] locations, int line, int index)

{

foreach(loc; locations)

{

if(spanContains(loc.span, line, index))

return loc;

}

return null;

}

if(lastLocOpen && stack.length > 0 && compareEndAddress(stack[$-1].span, line, index) <= 0)

return stack[$-1];

if(Location loc = findLocation(stack, line, index))

{

Location child;

while((child = findLocation(loc.children, line, index)) !is null)

loc = child;

return loc;

}

return null;

}

void writeTree(Location loc, int indent)

{

writeln(replicate(" ", indent), loc, " ", logString(loc.span));

foreach(child; loc.children)

writeTree(child, indent + 1);

}

void writeTree()

{

for(int i = 0; i < stack.length; i++)

{

writeln("Stack depth ", i);

writeTree(stack[i], 1);

}

}

void parseLine(ref int state, S line, int lno)

{

for(uint pos = 0; pos < line.length; )

{

ParserToken!S tok;

tok.span.iStartLine = lno;

tok.span.iStartIndex = pos;

tok.type = cast(TokenCat) spLex.scan(state, line, pos, tok.id);

tok.text = line[tok.span.iStartIndex .. pos];

if(pos == line.length)

{

// join end of line and beginning of next line

tok.span.iEndLine = lno + 1;

tok.span.iEndIndex = 0;

}

else

{

tok.span.iEndLine = lno;

tok.span.iEndIndex = pos;

}

if(!Lexer.isCommentOrSpace(tok.type, line[tok.span.iStartIndex .. $]))

shift(tok);

}

}

void OnLinesChanged(int iStartLine, int iOldEndLine, int iNewEndLine)

{

}

}

version(all) {}

else version(MAIN)

{

import parser.engine;

int main(string[] argv)

{

return 0;

}

}

else version(MAIN)

{

import dparser;

int main(string[] argv)

{

genDParser();

return 0;

}

}

else version(MAIN)

{

int main(string[] argv)

{

string text = q{

class A {

int x;

version(none)

int fn()

{

test;

}

else

int bar();

if(1)

if(2)

debug(3) a;

else b;

c;

}

};

auto parser = new ParserBase!string;

string[] lines = split(text, "\n");

int state = 0;

int[] states = new int[lines.length+1];

states[0] = state;

foreach(lno, line; lines)

{

parser.parseLine(state, line, lno);

states[lno+1] = state;

}

parser.writeTree();

assert(parser.stack.length == 1);

int line = 7, index = 0;

parser.prune(line, index);

parser.writeTree();

state = states[line];

for(int i = line; i < lines.length; i++)

parser.parseLine(state, lines[i], i);

parser.writeTree();

assert(parser.stack.length == 1);

auto verloc = parser.findLocation(4, 6, true);

assert(cast(VersionStatement!string) verloc);

return 0;

}

}

// This file is part of Visual D

//

// Visual D integrates the D programming language into Visual Studio

// Copyright (c) 2010 by Rainer Schuetze, All Rights Reserved

//

// Distributed under the Boost Software License, Version 1.0.

// See accompanying file LICENSE\_1\_0.txt or copy at <http://www.boost.org/LICENSE_1_0.txt>

module visuald.stringutil;

import visuald.windows;

import visuald.comutil;

import stdext.file;

import core.stdc.stdlib;

//import std.windows.charset;

import std.path;

import std.utf;

import std.string;

import std.ascii;

import std.conv;

import std.array;

string ellipseString(string s, int maxlen)

{

if (s.length > maxlen - 1)

s = s[0 .. maxlen - 4] ~ "...";

return s;

}

void addFileMacros(string path, string base, ref string[string] replacements)

{

replacements[base ~ "PATH"] = path;

replacements[base ~ "DIR"] = dirName(path);

string filename = baseName(path);

string ext = extension(path);

if(ext.startsWith("."))

ext = ext[1..$];

replacements[base ~ "FILENAME"] = filename;

replacements[base ~ "EXT"] = ext;

string name = stripExtension(filename);

replacements[base ~ "NAME"] = name.length == 0 ? filename : name;

}

string getEnvVar(string var)

{

wchar[256] wbuf;

const(wchar)\* wvar = toUTF16z(var);

uint cnt = GetEnvironmentVariable(wvar, wbuf.ptr, 256);

if(cnt < 256)

return to\_string(wbuf.ptr, cnt);

wchar[] pbuf = new wchar[cnt+1];

cnt = GetEnvironmentVariable(wvar, pbuf.ptr, cnt + 1);

return to\_string(pbuf.ptr, cnt);

}

string \_replaceMacros(string start, dchar end, string esc)(string s, string[string] replacements)

{

int[string] lastReplacePos;

auto slen = start.length;

for(int i = 0; i + slen < s.length; )

{

if(s[i .. i+esc.length] == esc)

s = s[0 .. i] ~ s[i + 1 .. $];

else if(s[i .. i+slen] == start)

{

int len = indexOf(s[i+slen .. $], end);

if(len < 0)

break;

string id = toUpper(s[i + slen .. i + slen + len]);

string nid;

if(string \*ps = id in replacements)

nid = \*ps;

else

nid = getEnvVar(id);

int \*p = id in lastReplacePos;

if(!p || \*p <= i)

{

s = s[0 .. i] ~ nid ~ s[i + slen + 1 + len .. $];

int difflen = nid.length - (len + slen + 1);

foreach(ref int pos; lastReplacePos)

if(pos > i)

pos += difflen;

lastReplacePos[id] = i + nid.length;

continue;

}

}

i++;

}

return s;

}

string replaceMacros(string s, string[string] replacements)

{

return \_replaceMacros!("$(", ')', "$$")(s, replacements);

}

string replaceEnvironment(string s, string[string] replacements)

{

return \_replaceMacros!("%", '%', "%%")(s, replacements);

}

// ATTENTION: env modified

string[string] expandIniSectionEnvironment(string txt, string[string] env)

{

string[2][] lines = parseIniSectionAssignments(txt);

foreach(ref record; lines)

{

string id = toUpper(record[0]);

string expr = record[1];

string val = replaceEnvironment(expr, env);

env[id] = val;

}

return env;

}

unittest

{

string[string] env = [ "V1" : "x1", "V2" : "x2" ];

string ini = `

i1 = i%v1%

; comment

i2 = %i1%\_i2

; comment with =

v2 = %v2%;i2`;

env = expandIniSectionEnvironment(ini, env);

//import std.stdio;

//writeln(env);

assert(env["I1"] == "ix1");

assert(env["I2"] == "ix1\_i2");

assert(env["V1"] == "x1");

assert(env["V2"] == "x2;i2");

}

S createPasteString(S)(S s)

{

S t;

bool wasWhite = false;

foreach(dchar ch; s)

{

if(t.length > 30)

return t ~ "...";

bool isw = isWhite(ch);

if(ch == '&')

t ~= "&&";

else if(!isw)

t ~= ch;

else if(!wasWhite)

t ~= ' ';

wasWhite = isw;

}

return t;

}

// special version of std.string.indexOf that considers '.', '/' and '\\' the same

// character for case insensitive searches

ptrdiff\_t indexOfPath(Char1, Char2)(const(Char1)[] s, const(Char2)[] sub,

std.string.CaseSensitive cs = std.string.CaseSensitive.yes)

{

const(Char1)[] balance;

if (cs == std.string.CaseSensitive.yes)

{

balance = std.algorithm.find(s, sub);

}

else

{

static bool isSame(Char1, Char2)(Char1 c1, Char2 c2)

{

if (c1 == '.' || c1 == '/' || c1 == '\\')

return c2 == '.' || c2 == '/' || c2 == '\\';

return std.uni.toLower(c1) == std.uni.toLower(c2);

}

balance = std.algorithm.find!((a, b) => isSame(a, b))(s, sub);

}

return balance.empty ? -1 : balance.ptr - s.ptr;

}

// This file is part of Visual D

//

// Visual D integrates the D programming language into Visual Studio

// Copyright (c) 2010-2011 by Rainer Schuetze, All Rights Reserved

//

// Distributed under the Boost Software License, Version 1.0.

// See accompanying file LICENSE\_1\_0.txt or copy at <http://www.boost.org/LICENSE_1_0.txt>

///////////////////////////////////////////////////////////////////////

//

// replace a series of tokens

//

// special items in search string (NAME can be any alpha numeric identifier):

// $\_numNAME - any integer literal

// $\_identNAME - any identifier (no keywords)

// $\_dotidentNAME - any identifier.identifier pair

// $\_exprNAME - any sequence of brace matched tokens terminated by closing bracket or ";"

// $\_notNAME - any token not matching the following token

// $\_optNAME - the following token or nothing

// $NAME - any sequence of tokens greedily stopped by the following token

// token$NAME - any token starting with "token"

//

// special items in the replacement string

// any $-names used in the replacement string

// $\* - the full matched string

//

///////////////////////////////////////////////////////////////////////

module visuald.tokenreplace;

import vdc.lexer;

import c2d.dlist;

import std.string;

import std.ascii;

import std.conv;

alias wstring \_string;

///////////////////////////////////////////////////////////////////////

private void throwException(int line, \_string msg)

{

if(line > 0)

throw new Exception(format("(%d):", line) ~ to!string(msg));

throw new Exception(to!string(msg));

}

///////////////////////////////////////////////////////////////////////

class Token

{

enum Comment = TOK\_Comment;

enum Newline = TOK\_Comment;

enum Identifier = TOK\_Identifier;

enum Number = TOK\_IntegerLiteral;

enum Dot = TOK\_dot;

enum EOF = TOK\_EOF;

enum ParenL = TOK\_lparen;

enum ParenR = TOK\_rparen;

enum BraceL = TOK\_lcurly;

enum BraceR = TOK\_rcurly;

enum BracketL = TOK\_lbracket;

enum BracketR = TOK\_rbracket;

static bool isPPToken(int) { return false; }

bool isOpeningBracket() { return type == ParenL || type == BraceL || type == BracketL; }

bool isClosingBracket() { return type == ParenR || type == BraceR || type == BracketR; }

int type;

bool replaced;

int lineno, column; // token pos and end can be calculated from pretext/text

\_string text;

\_string pretext;

}

alias DList!(Token) TokenList;

alias DListIterator!(Token) TokenIterator;

struct TokenRange

{

TokenIterator start;

TokenIterator end;

}

struct SubMatch

{

\_string ident;

TokenIterator start;

TokenIterator end;

}

struct ReplaceRange

{

// offsets into old and new \_string

int startlineno;

int startcolumn;

int endlineno;

int endcolumn;

\_string replacementText;

}

struct ReplaceOptions

{

bool matchCase = true;

bool matchBrackets = true;

bool keepCase = true;

bool includePretext = false;

bool findOnly = false;

bool findMultiple = false;

}

//////////////////////////////////////////////////////////////////////////////

\_\_gshared Lexer trLex;

shared static this()

{

trLex.mAllowDollarInIdentifiers = true;

}

//////////////////////////////////////////////////////////////////////////////

void advanceTextPos(\_string text, ref int lineno, ref int column)

{

for( ; ; )

{

int pos = indexOf(text, '\n');

if(pos < 0)

break;

lineno++;

column = 0;

text = text[pos+1 .. $];

}

column += text.length;

}

//////////////////////////////////////////////////////////////////////////////

Token createToken(\_string pretext, \_string text, int type, int lineno, int column)

{

Token tok = new Token();

tok.pretext = pretext;

tok.text = text;

tok.type = type;

tok.lineno = lineno;

tok.column = column;

return tok;

}

Token createToken(Token tok)

{

Token ntok = new Token();

ntok.pretext = tok.pretext;

ntok.text = tok.text;

ntok.type = tok.type;

ntok.lineno = tok.lineno;

ntok.column = tok.column;

return ntok;

}

bool isCommentToken(Token tok, bool checkPP = true)

{

return tok.type == Token.Comment || tok.type == Token.Newline || (checkPP && Token.isPPToken(tok.type));

}

void skipComments(ref TokenIterator tokIt, bool skipPP = true)

{

while (!tokIt.atEnd() && isCommentToken(\*tokIt, skipPP))

tokIt.advance();

}

void nextToken(ref TokenIterator tokIt, bool skipPP = true)

{

tokIt.advance();

skipComments(tokIt, skipPP);

}

\_string tokensToIdentifier(TokenIterator start, TokenIterator end)

{

\_string ident;

while(!start.atEnd() && start != end)

{

if(ident.length > 0 && start.text.length > 0)

if(isAlphaNum(ident[$-1]) && isAlphaNum(start.text[0]))

ident ~= " ";

ident ~= start.text;

++start;

}

return ident;

}

//////////////////////////////////////////////////////////////////////////////

TokenList copyTokenList(TokenIterator start, TokenIterator end, bool cloneTokens = true)

{

TokenList tokenList = new TokenList;

for(TokenIterator it = start; it != end; ++it)

{

Token tok = cloneTokens ? createToken(\*it) : \*it;

tokenList.append(tok);

}

return tokenList;

}

TokenList copyTokenList(TokenRange range, bool cloneTokens = true)

{

return copyTokenList(range.start, range.end, cloneTokens);

}

TokenList copyTokenList(TokenList tokenList, bool cloneTokens = true)

{

return copyTokenList(tokenList.begin(), tokenList.end(), cloneTokens);

}

TokenIterator insertTokenList(TokenIterator insBefore, TokenList tokenList)

{

if(tokenList.empty())

return insBefore;

TokenIterator endit = tokenList.end() - 1;

if(endit.type == Token.EOF && !insBefore.atEnd())

{

insBefore.pretext = endit.pretext ~ insBefore.pretext;

endit.erase();

}

return insBefore.insertListBefore(tokenList);

}

\_string tokenListToString(TokenIterator start, TokenIterator end, bool checkSpaceBetweenIdentifiers = false,

bool normalizePreText = false)

{

\_string text;

\_string prevtext;

for(TokenIterator tokIt = start; tokIt != end; ++tokIt)

{

Token tok = \*tokIt;

\_string txt = normalizePreText ? tok.text : tok.pretext ~ tok.text;

if(checkSpaceBetweenIdentifiers || normalizePreText)

{

if (prevtext == "\_\_")

txt = tok.text;

else if (tok.text == "\_\_")

txt = "";

else if (txt.length && prevtext.length)

{

dchar prevch = prevtext[$-1];

dchar ch = txt[0];

if((isAlphaNum(ch) || ch == '\_') && (isAlphaNum(prevch) || prevch == '\_'))

txt = " " ~ txt;

}

prevtext = tok.text;

}

text ~= txt;

}

return text;

}

void markReplaceTokenList(TokenIterator start, TokenIterator end, bool replaced = true)

{

for(TokenIterator it = start; it != end; ++it)

it.replaced = replaced;

}

void markReplaceTokenList(TokenList tokenList, bool replaced = true)

{

markReplaceTokenList(tokenList.begin(), tokenList.end(), replaced);

}

\_string tokenListToString(TokenList tokenList, bool checkSpaceBetweenIdentifiers = false)

{

return tokenListToString(tokenList.begin(), tokenList.end(), checkSpaceBetweenIdentifiers);

}

bool compareTokenList(TokenIterator start1, TokenIterator end1, TokenIterator start2, TokenIterator end2)

{

TokenIterator it1 = start1;

TokenIterator it2 = start2;

for( ; it1 != end1 && it2 != end2; ++it1, ++it2)

if(it1.text != it2.text)

return false;

return it1 == end1 && it2 == end2;

}

//////////////////////////////////////////////////////////////////////////////

// iterator on token after closing bracket

bool advanceToClosingBracket(ref TokenIterator it, TokenIterator stopIt)

{

TokenIterator prevIt = it; // for debugging

int lineno = it.lineno;

int open = it.type;

int close;

switch(open)

{

case Token.ParenL:

close = Token.ParenR;

break;

case Token.BraceL:

close = Token.BraceR;

break;

case Token.BracketL:

close = Token.BracketR;

break;

default:

throwException(lineno, "opening bracket expected instead of " ~ it.text);

}

int level = 1;

++it;

while (level > 0)

{

if(it == stopIt)

return false;

if(it.atEnd())

throwException(lineno, "end of file while looking for closing bracket");

if(it.type == open)

level++;

else if(it.type == close)

level--;

++it;

}

return true;

}

bool advanceToClosingBracket(ref TokenIterator it)

{

TokenIterator noStop;

return advanceToClosingBracket(it, noStop);

}

// iterator on token with opening bracket

bool retreatToOpeningBracket(ref TokenIterator it, TokenIterator stopIt)

{

int lineno = it.lineno;

int open;

int close = it.type;

switch(close)

{

case Token.ParenR:

open = Token.ParenL;

break;

case Token.BraceR:

open = Token.BraceL;

break;

case Token.BracketR:

open = Token.BracketL;

break;

default:

throwException(lineno, "closing bracket expected instead of " ~ it.text);

}

int level = 1;

while (level > 0)

{

--it;

if(it == stopIt)

return false;

if(it.atEnd())

throwException(lineno, "beginnig of file while looking for opening bracket");

if(it.type == close)

level++;

else if(it.type == open)

level--;

}

return true;

}

bool retreatToOpeningBracket(ref TokenIterator it)

{

TokenIterator noStop;

return retreatToOpeningBracket(it, noStop);

}

//////////////////////////////////////////////////////////////////////////////

static void scanAny(TL)(ref TL tokenList, \_string text, int lineno = 1, int column = 0, bool combinePP = true)

{

uint lastTokEnd = 0;

int state = 0;

int prelineno = lineno;

int precolumn = column;

void appendToken(Token tok)

{

static if(is(TL == Token[]))

tokenList ~= tok;

else static if(is(TL == \_string[]))

tokenList ~= tok.text;

else

tokenList.append(tok);

}

for(uint pos = 0; pos < text.length; )

{

int tokid;

uint prevpos = pos;

trLex.scan(state, text, pos, tokid);

\_string txt = text[prevpos .. pos];

advanceTextPos(txt, lineno, column);

if(tokid != TOK\_Space && tokid != TOK\_Comment)

{

\_string pretext = text[lastTokEnd .. prevpos];

lastTokEnd = pos;

Token tok = createToken(pretext, txt, tokid, prelineno, precolumn);

appendToken(tok);

prelineno = lineno;

precolumn = column;

}

}

if(lastTokEnd < text.length)

{

\_string pretext = text[lastTokEnd .. $];

Token tok = createToken(pretext, text[$ .. $], TOK\_EOF, prelineno, precolumn);

appendToken(tok);

}

}

TokenList scanText(\_string text, int lineno = 1, int column = 0, bool combinePP = true)

{

TokenList tokenList = new TokenList;

scanAny(tokenList, text, lineno, column, combinePP);

return tokenList;

}

void scanTextArray(TYPE)(ref TYPE[] tokens, \_string text, int lineno = 1, int column = 0, bool combinePP = true)

{

scanAny(tokens, text, lineno, column, combinePP);

static if(is(TYPE == \_string))

{

while(tokens.length > 0 && tokens[$-1].length == 0)

tokens = tokens[0..$-1];

}

else

{

while(tokens.length > 0 && tokens[$-1].text.length == 0)

tokens = tokens[0..$-1];

}

}

///////////////////////////////////////////////////////////////////////

int findSubmatch(ref SubMatch[] submatch, \_string ident)

{

for(int i = 0; i < submatch.length; i++)

if(submatch[i].ident == ident)

return i;

return -1;

}

///////////////////////////////////////////////////////////////////////

bool findTokenSequence(TokenIterator it, \_string[] search,

bool checkBracketsSearch, bool checkBracketsMatch, bool caseSensitive,

\_string stopText, ref TokenRange match, ref SubMatch[] submatch)

{

if(search.length == 0)

{

match.start = it;

match.end = it;

return true;

}

void addSubmatch(\_string search, TokenIterator start, TokenIterator end)

{

SubMatch smatch;

smatch.ident = search;

smatch.start = start;

smatch.end = end;

submatch ~= smatch;

}

bool strEqual(\_string s1, \_string s2)

{

if(caseSensitive)

return s1 == s2;

return icmp(s1, s2) == 0;

}

bool compareTokens(TokenIterator start, TokenIterator end, ref TokenIterator it)

{

for(TokenIterator sit = start; !sit.atEnd() && sit != end; ++sit)

{

\_string sittext = strip(sit.text);

if(sittext.length == 0)

continue;

while(!it.atEnd() && strip(it.text).length == 0)

++it;

if(it.atEnd())

return false;

if(!strEqual(strip(it.text), sittext))

return false;

++it;

}

return true;

}

bool compareSubmatch(ref SubMatch sm, \_string txt)

{

\_string s = tokenListToString(sm.start, sm.end);

return strEqual(strip(s), strip(txt));

}

int p = 0;

while(p < search.length && search[p].length == 0)

p++;

if(p >= search.length)

return false;

int prevsubmatchLength = submatch.length;

while(!it.atEnd() && (stopText.length == 0 || !strEqual(it.text, stopText)

|| strEqual(search[p], stopText)))

{

bool dollar = indexOf(search[p], '$') >= 0;

if(strEqual(strip(it.text), search[p]) || dollar)

{

TokenIterator mit = it + (dollar ? 0 : 1);

int i = p + (dollar ? 0 : 1);

while(i < search.length && search[i].length == 0)

i++;

while(!mit.atEnd() && i < search.length)

{

\_string mittext = strip(mit.text);

if(mittext.length == 0)

{

++mit;

continue;

}

if(startsWith(search[i], "$"))

{

int idx = findSubmatch(submatch, search[i]);

if(idx >= 0)

{

if(!compareTokens(submatch[idx].start, submatch[idx].end, mit))

goto Lnomatch;

goto LnoAdvance;

}

else if(startsWith(search[i], "$\_num"))

{

if(mit.type != Token.Number)

break;

addSubmatch(search[i], mit, mit + 1);

}

else if(startsWith(search[i], "$\_ident"))

{

if(mit.type != Token.Identifier)

break;

addSubmatch(search[i], mit, mit + 1);

}

else if(startsWith(search[i], "$\_dotident"))

{

if(mit.type != Token.Identifier)

break;

TokenIterator start = mit;

while(!(mit + 1).atEnd() && !(mit + 2).atEnd() &&

mit[1].type == Token.Dot && mit[2].type == Token.Identifier)

{

mit.advance();

mit.advance();

}

addSubmatch(search[i], start, mit + 1);

}

else if(startsWith(search[i], "$\_expr"))

{

// ok to allow empty expression?

TokenRange tailmatch;

if (!findTokenSequence(mit, search[i+1 .. $], true, true, caseSensitive,

";", tailmatch, submatch))

break;

addSubmatch(search[i], mit, tailmatch.start);

mit = tailmatch.end;

i = search.length;

break;

}

else if(startsWith(search[i], "$\_not") && i + 1 < search.length)

{

if(startsWith(search[i + 1], "$\_ident"))

{

if(mit.type == Token.Identifier)

break;

}

else if(startsWith(search[i + 1], "$\_num"))

{

if(mit.type == Token.Number)

break;

}

else if(strEqual(mittext, search[i + 1]))

break;

addSubmatch(search[i], mit, mit + 1);

i++;

}

else if(startsWith(search[i], "$\_opt"))

{

i++;

if(i < search.length && strEqual(mittext, search[i]))

addSubmatch(search[i-1], mit, mit + 1);

else

{

addSubmatch(search[i-1], mit, mit);

goto LnoAdvance; // nothing matched

}

}

else

{

TokenRange tailmatch;

if (!findTokenSequence(mit, search[i+1 .. $],

checkBracketsMatch, checkBracketsMatch, caseSensitive,

stopText, tailmatch, submatch))

break;

addSubmatch(search[i], mit, tailmatch.start);

mit = tailmatch.end;

i = search.length;

break;

}

}

else

{

int idx = indexOf(search[i], '$');

if(idx < 0)

{

if (!strEqual(mittext, search[i]))

break;

}

else if(mittext.length < idx)

break;

else if(!strEqual(mittext[0 .. idx], search[i][0 .. idx]))

break;

else

{

int sidx = findSubmatch(submatch, search[i][idx .. $]);

if(sidx < 0)

{

// create dummy token and list to add a submatch

Token subtok = createToken("", mittext[idx .. $], Token.Identifier, mit.lineno, mit.column);

TokenList sublist = new TokenList;

sublist.append(subtok);

addSubmatch(search[i][idx .. $], sublist.begin(), sublist.end());

}

else if(!compareSubmatch(submatch[sidx], mittext[idx .. $]))

break;

}

}

++mit;

LnoAdvance:

i++;

while(i < search.length && search[i].length == 0)

i++;

}

if(i >= search.length)

{

match.start = it;

match.end = mit;

return true;

}

Lnomatch:

submatch.length = prevsubmatchLength;

}

if(checkBracketsSearch && it.isOpeningBracket())

advanceToClosingBracket(it);

else if(checkBracketsSearch && it.isClosingBracket())

break;

else

it.advance();

}

return false;

}

TokenList createReplacementTokenList(RTYPE) (RTYPE[] replace, TokenRange match, ref SubMatch[] submatch)

{

TokenList tokenList = new TokenList;

for(int i = 0; i < replace.length; i++)

{

\_string reptext;

\_string pretext;

int type = Token.Comment;

static if (is(RTYPE == Token))

{

reptext = replace[i].text;

pretext = replace[i].pretext;

type = replace[i].type;

if(reptext == "$" && i + 1 < replace.length && replace[i+1].pretext == "")

{

reptext ~= replace[i + 1].text;

i++;

}

}

else

{

reptext = replace[i];

}

if(reptext == "$\*")

tokenList.appendList(copyTokenList(match));

else if(startsWith(reptext, "$"))

{

int idx = findSubmatch(submatch, reptext);

if(idx < 0)

throwException(0, "no submatch for " ~ reptext);

TokenList list = copyTokenList(submatch[idx].start, submatch[idx].end);

if(!list.empty() && pretext.length)

list.begin().pretext = pretext ~ list.begin().pretext;

tokenList.appendList(list);

}

else

{

Token tok = createToken(pretext, reptext, type, 0, 0);

tokenList.append(tok);

}

}

return tokenList;

}

int \_replaceTokenSequence(RTYPE)(TokenList srctoken, \_string[] search, RTYPE[] replace,

ref const ReplaceOptions opt, ReplaceRange[]\* ranges)

{

if(search.length == 0) // do not replace an empty token list (everything?)

return 0;

for(int i = 0; i < search.length; i++)

search[i] = strip(search[i]);

int cntReplacements = 0;

TokenIterator it = srctoken.begin();

for( ; ; )

{

TokenRange match;

SubMatch[] submatch;

if(!findTokenSequence(it, search, false, opt.matchBrackets, opt.matchCase, "", match, submatch))

break;

ReplaceRange rng;

if(ranges)

{

if(match.end.atEnd())

{

TokenIterator mit = match.end - 1;

rng.endlineno = mit.lineno;

rng.endcolumn = mit.column;

advanceTextPos(mit.pretext, rng.endlineno, rng.endcolumn);

advanceTextPos(mit.text, rng.endlineno, rng.endcolumn);

}

else

{

rng.endlineno = match.end.lineno;

rng.endcolumn = match.end.column;

}

}

if(!opt.findOnly)

{

\_string pretext;

if(!opt.includePretext)

{

pretext = match.start.pretext;

match.start.pretext = "";

advanceTextPos(pretext, match.start.lineno, match.start.column);

}

TokenList tokenList = createReplacementTokenList(replace, match, submatch);

markReplaceTokenList(tokenList);

if(ranges)

{

rng.startlineno = match.start.lineno;

rng.startcolumn = match.start.column;

rng.replacementText = tokenListToString(tokenList);

\*ranges ~= rng;

}

if(!tokenList.empty())

tokenList.begin().pretext = pretext ~ tokenList.begin().pretext;

srctoken.remove(match.start, match.end);

srctoken.insertListBefore(match.end, tokenList);

}

else

{

if(ranges)

{

rng.startlineno = match.start.lineno;

rng.startcolumn = match.start.column;

if(!opt.includePretext)

advanceTextPos(match.start.pretext, rng.startlineno, rng.startcolumn);

\*ranges ~= rng;

}

if(!opt.findMultiple)

return 1;

}

it = match.end; // should we recurse into the replacement?

cntReplacements++;

}

return cntReplacements;

}

int replaceTokenSequence(TokenList srctoken, \_string[] search, \_string[] replace,

ref const ReplaceOptions opt, ReplaceRange[]\* ranges)

{

return \_replaceTokenSequence(srctoken, search, replace, opt, ranges);

}

int replaceTokenSequence(TokenList srctoken, \_string search, \_string replace,

ref const ReplaceOptions opt, ReplaceRange[]\* ranges)

{

\_string[] searchTokens;

scanTextArray!(\_string)(searchTokens, search);

Token[] replaceTokens;

scanTextArray!(Token)(replaceTokens, replace);

return \_replaceTokenSequence(srctoken, searchTokens, replaceTokens, opt, ranges);

}

\_string replaceTokenSequence(\_string srctext, int srclineno, int srccolumn, \_string search, \_string replace,

ref const ReplaceOptions opt, ReplaceRange[]\* ranges)

{

TokenList tokens = scanText(srctext, srclineno, srccolumn);

int cnt = replaceTokenSequence(tokens, search, replace, opt, ranges);

if(cnt == 0)

return srctext;

\_string newtext = tokenListToString(tokens);

return newtext;

}

unittest

{

\_string txt =

"unittest {\n"

" if (list\_freelist) {\n"

" list--;\n"

" }\n"

"}\n"

;

ReplaceOptions opt;

ReplaceRange[] rng1;

\_string res1 = replaceTokenSequence(txt, 1, 0, "if($1) { $2 }", "$2", opt, &rng1);

\_string exp1 =

"unittest {\n"

" \n"

" list--;\n"

"}\n"

;

assert(res1 == exp1);

assert(rng1.length == 1);

assert(rng1[0].startlineno == 2 && rng1[0].startcolumn == 2);

assert(rng1[0].endlineno == 4 && rng1[0].endcolumn == 3);

opt.includePretext = true;

ReplaceRange[] rng2;

\_string res2 = replaceTokenSequence(txt, 1, 0, "if($1) { $2 }", "$2", opt, &rng2);

\_string exp2 =

"unittest {\n"

" list--;\n"

"}\n"

;

assert(res2 == exp2);

assert(rng2.length == 1);

assert(rng2[0].startlineno == 1 && rng2[0].startcolumn == 10);

assert(rng2[0].endlineno == 4 && rng2[0].endcolumn == 3);

}

// This file is part of Visual D

//

// Visual D integrates the D programming language into Visual Studio

// Copyright (c) 2010-2011 by Rainer Schuetze, All Rights Reserved

//

// Distributed under the Boost Software License, Version 1.0.

// See accompanying file LICENSE\_1\_0.txt or copy at <http://www.boost.org/LICENSE_1_0.txt>

module visuald.tokenreplacedialog;

import visuald.windows;

import visuald.winctrl;

import visuald.comutil;

import visuald.logutil;

import visuald.hierutil;

import visuald.stringutil;

import visuald.pkgutil;

import visuald.wmmsg;

import visuald.dpackage;

import visuald.dimagelist;

import visuald.tokenreplace;

import visuald.register;

import sdk.win32.commctrl;

import sdk.vsi.vsshell;

import sdk.vsi.vsshell80;

import dte80a = sdk.vsi.dte80a;

import dte80 = sdk.vsi.dte80;

import std.algorithm;

import std.conv;

private IVsWindowFrame sWindowFrame;

private        TokenReplacePane sSearchPane;

const int kPaneMargin = 0;

const int kBackMargin = 4;

bool showTokenReplaceWindow(bool replace)

{

if(!sWindowFrame)

{

auto pIVsUIShell = ComPtr!(IVsUIShell)(queryService!(IVsUIShell), false);

if(!pIVsUIShell)

return false;

sSearchPane = newCom!TokenReplacePane();

const(wchar)\* caption = "Visual D Token Search/Replace"w.ptr;

HRESULT hr;

hr = pIVsUIShell.CreateToolWindow(CTW\_fInitNew, 0, sSearchPane,

&GUID\_NULL, &g\_tokenReplaceWinCLSID, &GUID\_NULL,

null, caption, null, &sWindowFrame);

if(!SUCCEEDED(hr))

{

sSearchPane = null;

return false;

}

}

if(FAILED(sWindowFrame.Show()))

return false;

BOOL fHandled;

sSearchPane.\_OnSetFocus(0, 0, 0, fHandled);

return fHandled != 0;

}

bool findNextTokenReplace(bool up)

{

if(!sSearchPane)

return false;

return sSearchPane.\_DoFindNext(up) == 0;

}

bool closeTokenReplaceWindow()

{

sWindowFrame = release(sWindowFrame);

sSearchPane = null;

return true;

}

class TokenReplaceWindowBack : Dialog

{

this(Window parent, TokenReplacePane pane)

{

mPane = pane;

super(parent);

}

override int WindowProc(HWND hWnd, uint uMsg, WPARAM wParam, LPARAM lParam)

{

BOOL fHandled;

LRESULT rc = mPane.\_WindowProc(hWnd, uMsg, wParam, lParam, fHandled);

if(fHandled)

return rc;

return super.WindowProc(hWnd, uMsg, wParam, lParam);

}

TokenReplacePane mPane;

}

class TokenReplacePane : DisposingComObject, IVsWindowPane

{

IServiceProvider mSite;

override HRESULT QueryInterface(in IID\* riid, void\*\* pvObject)

{

if(queryInterface!(IVsWindowPane) (this, riid, pvObject))

return S\_OK;

// avoid debug output

if(\*riid == IVsCodeWindow.iid || \*riid == IServiceProvider.iid || \*riid == IVsTextView.iid)

return E\_NOINTERFACE;

return super.QueryInterface(riid, pvObject);

}

override void Dispose()

{

mSite = release(mSite);

}

HRESULT SetSite(/+[in]+/ IServiceProvider pSP)

{

mixin(LogCallMix2);

mSite = release(mSite);

mSite = addref(pSP);

return S\_OK;

}

HRESULT CreatePaneWindow(in HWND hwndParent, in int x, in int y, in int cx, in int cy,

/+[out]+/ HWND \*hwnd)

{

mixin(LogCallMix2);

\_wndParent = new Window(hwndParent);

\_wndBack = new TokenReplaceWindowBack(\_wndParent, this);

BOOL fHandled;

\_OnInitDialog(WM\_INITDIALOG, 0, 0, fHandled);

\_CheckSize();

\_wndBack.setVisible(true);

return S\_OK;

}

HRESULT GetDefaultSize(/+[out]+/ SIZE \*psize)

{

mixin(LogCallMix2);

psize.cx = 300;

psize.cy = 200;

return S\_OK;

}

HRESULT ClosePane()

{

mixin(LogCallMix2);

if(\_wndParent)

{

\_WriteStateToRegistry();

\_wndParent.Dispose();

\_wndParent = null;

\_wndBack = null;

\_wndToolbar = null;

\_wndFindLabel = null;

\_wndFindText = null;

\_wndReplaceLabel = null;

\_wndReplaceText = null;

\_wndMatchCase = null;

\_wndMatchBraces = null;

\_wndIncComment = null;

\_wndReplaceCase = null;

\_wndDirectionUp = null;

\_wndLookInLabel = null;

\_wndLookIn = null;

\_wndNext = null;

\_wndReplace = null;

\_wndReplaceAll = null;

\_wndClose = null;

if(\_himlToolbar)

ImageList\_Destroy(\_himlToolbar);

mDlgFont = deleteDialogFont(mDlgFont);

}

return S\_OK;

}

HRESULT LoadViewState(/+[in]+/ IStream pstream)

{

mixin(LogCallMix2);

return returnError(E\_NOTIMPL);

}

HRESULT SaveViewState(/+[in]+/ IStream pstream)

{

mixin(LogCallMix2);

return returnError(E\_NOTIMPL);

}

HRESULT TranslateAccelerator(MSG\* msg)

{

if(msg.message == WM\_TIMER)

\_CheckSize();

if(msg.message == WM\_TIMER || msg.message == WM\_SYSTIMER)

return E\_NOTIMPL; // do not flood debug output

logMessage("TranslateAccelerator", msg.hwnd, msg.message, msg.wParam, msg.lParam);

BOOL fHandled;

HRESULT hrRet = \_HandleMessage(msg.hwnd, msg.message, msg.wParam, msg.lParam, fHandled);

if(fHandled)

return hrRet;

return E\_NOTIMPL;

}

///////////////////////////////////////////////////////////////////

// the following has been ported from the FlatSolutionExplorer project

private:

Window \_wndParent;

TokenReplaceWindowBack \_wndBack;

ToolBar \_wndToolbar;

HIMAGELIST \_himlToolbar;

ReplaceOptions \_options;

HFONT mDlgFont;

Label \_wndFindLabel;

MultiLineText \_wndFindText;

Label \_wndReplaceLabel;

MultiLineText \_wndReplaceText;

CheckBox \_wndMatchCase;

CheckBox \_wndMatchBraces;

CheckBox \_wndIncComment;

CheckBox \_wndReplaceCase;

CheckBox \_wndDirectionUp;

Label \_wndLookInLabel;

ComboBox \_wndLookIn;

Button \_wndNext;

Button \_wndReplace;

Button \_wndReplaceAll;

Button \_wndClose;

static HINSTANCE getInstance() { return Widget.getInstance(); }

int \_WindowProc(HWND hWnd, uint uMsg, WPARAM wParam, LPARAM lParam, ref BOOL fHandled)

{

if(uMsg != WM\_NOTIFY)

logMessage("\_WindowProc", hWnd, uMsg, wParam, lParam);

return \_HandleMessage(hWnd, uMsg, wParam, lParam, fHandled);

}

int \_HandleMessage(HWND hWnd, uint uMsg, WPARAM wParam, LPARAM lParam, ref BOOL fHandled)

{

switch(uMsg)

{

case WM\_CREATE:

case WM\_INITDIALOG:

return \_OnInitDialog(uMsg, wParam, lParam, fHandled);

case WM\_DESTROY:

return \_OnDestroy(uMsg, wParam, lParam, fHandled);

case WM\_SIZE:

if(hWnd == \_wndBack.hwnd)

return \_OnSize(uMsg, wParam, lParam, fHandled);

break;

case WM\_KEYDOWN:

case WM\_SYSKEYDOWN:

return \_OnKeyDown(uMsg, wParam, lParam, fHandled);

case WM\_NCACTIVATE:

case WM\_SETFOCUS:

return \_OnSetFocus(uMsg, wParam, lParam, fHandled);

case WM\_COMMAND:

ushort id = LOWORD(wParam);

ushort code = HIWORD(wParam);

//                        if(id == IDC\_FINDTEXT && code == EN\_CHANGE)

//                                return \_OnFileWheelChanged(id, code, hWnd, fHandled);

if(code == BN\_CLICKED)

{

switch(id)

{

case IDC\_FINDCLOSE:

sWindowFrame.Hide();

return 0;

case IDC\_FINDNEXT:

return \_OnFindNext();

case IDC\_REPLACE:

return \_OnReplace();

case IDC\_REPLACEALL:

return \_OnReplaceAll();

default:

break;

}

}

break;

/+

case WM\_NCCALCSIZE:

return \_OnCalcSize(uMsg, wParam, lParam, fHandled);

case WM\_CONTEXTMENU:

return \_OnContextMenu(uMsg, wParam, lParam, fHandled);

case WM\_NOTIFY:

if (nmhdr.idFrom == IDC\_TOOLBAR && nmhdr.code == TBN\_GETINFOTIP)

return \_OnToolbarGetInfoTip(wParam, nmhdr, fHandled);

break;

+/

default:

break;

}

return 0;

}

LRESULT \_OnInitDialog(UINT uiMsg, WPARAM wParam, LPARAM lParam, ref BOOL fHandled)

{

if(\_wndFindLabel)

return S\_OK;

updateEnvironmentFont();

if(!mDlgFont)

mDlgFont = newDialogFont();

\_wndFindLabel = new Label(\_wndBack, "Fi&nd what:", -1);

\_wndFindText = new MultiLineText(\_wndBack, "", IDC\_FINDTEXT);

\_wndReplaceLabel = new Label(\_wndBack, "Re&place with:", -1);

\_wndReplaceText = new MultiLineText(\_wndBack, "", IDC\_REPLACETEXT);

\_wndMatchCase = new CheckBox(\_wndBack, "Match &case", IDC\_FINDMATCHCASE);

\_wndMatchBraces = new CheckBox(\_wndBack, "Match &braces", IDC\_FINDMATCHBRACES);

\_wndIncComment = new CheckBox(\_wndBack, "&Include preceding spaces and comments", IDC\_FINDINCCOMMENT);

\_wndDirectionUp = new CheckBox(\_wndBack, "Search &up", IDC\_FINDDIRECTION);

\_wndReplaceCase = new CheckBox(\_wndBack, "&Keep Case", IDC\_REPLACECASE);

\_wndLookInLabel = new Label(\_wndBack, "&Look in:", -1);

\_wndLookIn = new ComboBox(\_wndBack, [ "Current Document", "Current Selection"

/\*, "Current Project", "Current Solution"\*/ ], false, IDC\_FINDLOOKIN);

\_wndNext = new Button(\_wndBack, "&Find Next", IDC\_FINDNEXT);

\_wndReplace = new Button(\_wndBack, "&Replace", IDC\_REPLACE);

\_wndReplaceAll = new Button(\_wndBack, "Replace &All", IDC\_REPLACEALL);

\_wndClose = new Button(\_wndBack, "Close", IDC\_FINDCLOSE);

\_wndMatchCase .AddWindowStyle(WS\_TABSTOP);

\_wndMatchBraces.AddWindowStyle(WS\_TABSTOP);

\_wndIncComment .AddWindowStyle(WS\_TABSTOP);

\_wndDirectionUp.AddWindowStyle(WS\_TABSTOP);

\_ReadStateFromRegistry();

RECT r;

\_wndBack.GetClientRect(&r);

\_layoutViews(r.right - r.left, r.bottom - r.top);

// \_InitializeToolbar();

return S\_OK;

}

LRESULT \_OnDestroy(UINT uiMsg, WPARAM wParam, LPARAM lParam, ref BOOL fHandled)

{

if (\_himlToolbar)

{

\_wndToolbar.SendMessage(TB\_SETIMAGELIST, 0, cast(LPARAM)null);

ImageList\_Destroy(\_himlToolbar);

\_himlToolbar = null;

}

fHandled = TRUE;

// return CComCompositeControl<CFlatSolutionExplorer>::OnDestroy(uiMsg, wParam, lParam, fHandled);

return 0;

}

LRESULT \_OnKeyDown(UINT uiMsg, WPARAM wParam, LPARAM lParam, ref BOOL fHandled)

{

UINT vKey = LOWORD(wParam);

switch(vKey)

{

case VK\_ESCAPE:

sWindowFrame.Hide();

break;

default:

break;

}

return 0;

}

void \_CheckSize()

{

RECT r, br;

\_wndParent.GetClientRect(&r);

\_wndBack.GetClientRect(&br);

if(br.right - br.left != r.right - r.left - 2\*kPaneMargin ||

br.bottom - br.top != r.bottom - r.top - 2\*kPaneMargin)

\_wndBack.setRect(kPaneMargin, kPaneMargin,

r.right - r.left - 2\*kPaneMargin, r.bottom - r.top - 2\*kPaneMargin);

}

// Adjust child control sizes

void \_layoutViews(int cw, int ch)

{

int top = kBackMargin; // kToolBarAtTop ? kToolBarHeight : 1;

int bot = ch - kBackMargin;

int lineh = 16;

int combh = 20;

int lblspacing = 1;

int spacing = 3;

int btnw = 80;

int btnh = 24;

int x = kBackMargin;

int w = cw - 2 \* kBackMargin;

\_wndReplaceAll .setRect(x + w - btnw, bot - btnh, btnw, btnh);

\_wndReplace .setRect(x + w - btnw - spacing - btnw, bot - btnh, btnw, btnh);

bot -= btnh + spacing;

\_wndClose .setRect(x + w - btnw, bot - btnh, btnw, btnh);

\_wndNext .setRect(x + w - btnw - spacing - btnw, bot - btnh, btnw, btnh);

bot -= btnh + spacing + spacing;

\_wndLookIn .setRect(x, bot - combh, w, combh); bot -= combh + lblspacing;

\_wndLookInLabel .setRect(x, bot - lineh, w, lineh); bot -= lineh + spacing;

version(none)

{

\_wndReplaceCase .setRect(x, bot - lineh, w, lineh); bot -= lineh + spacing;

}

else

{

\_wndReplaceCase.setVisible(false);

}

\_wndDirectionUp .setRect(x + 100, bot - lineh, w - 100, lineh); // bot -= lineh + spacing;

\_wndMatchBraces .setRect(x, bot - lineh, 100, lineh); bot -= lineh + spacing;

\_wndIncComment .setRect(x + 100, bot - lineh, w - 100, lineh); // bot -= lineh + spacing;

\_wndMatchCase .setRect(x, bot - lineh, 100, lineh); bot -= lineh + spacing;

\_wndFindLabel .setRect(x, top, w, lineh); top += lineh + lblspacing;

int th = max(0, bot - top - spacing - lineh - spacing) / 2;

\_wndFindText .setRect(x, top, w, th); top += th + spacing;

\_wndReplaceLabel.setRect(x, top, w, lineh); top += lineh + lblspacing;

\_wndReplaceText .setRect(x, top, w, bot - top);

}

LRESULT \_OnSize(UINT uiMsg, WPARAM wParam, LPARAM lParam, ref BOOL fHandled)

{

int cx = LOWORD(lParam);

int cy = HIWORD(lParam);

\_layoutViews(cx, cy);

return 0;

}

LRESULT \_OnSetFocus(UINT uiMsg, WPARAM wParam, LPARAM lParam, ref BOOL fHandled)

{

// Skip the CComCompositeControl handling

// CComControl<CFlatSolutionExplorer, CAxDialogImpl<CFlatSolutionExplorer>>::OnSetFocus(uiMsg, wParam, lParam, fHandled);

if(\_wndFindText)

{

\_wndFindText.SetFocus();

\_wndFindText.SendMessage(EM\_SETSEL, 0, cast(LPARAM)-1);

fHandled = TRUE;

}

return 0;

}

void \_OptionsToDialog()

{

\_wndReplaceCase .setChecked(\_options.keepCase);

\_wndIncComment .setChecked(\_options.includePretext);

\_wndMatchBraces .setChecked(\_options.matchBrackets);

\_wndMatchCase .setChecked(\_options.matchCase);

}

void \_DialogToOptions()

{

\_options.keepCase = \_wndReplaceCase .isChecked();

//\_wndDirectionUp

\_options.includePretext = \_wndIncComment .isChecked();

\_options.matchBrackets = \_wndMatchBraces .isChecked();

\_options.matchCase = \_wndMatchCase .isChecked();

}

RegKey \_GetCurrentRegKey(bool write)

{

GlobalOptions opt = Package.GetGlobalOptions();

opt.getRegistryRoot();

wstring regPath = opt.regUserRoot ~ regPathToolsOptions;

regPath ~= "\\TokenReplaceWindow"w;

return new RegKey(opt.hUserKey, regPath, write);

}

bool \_WriteStateToRegistry()

{

try

{

\_DialogToOptions();

scope RegKey keyWinOpts = \_GetCurrentRegKey(true);

keyWinOpts.Set("keepCase"w, \_options.keepCase);

keyWinOpts.Set("matchCase"w, \_options.matchCase);

keyWinOpts.Set("includePretext"w, \_options.includePretext);

keyWinOpts.Set("matchBrackets"w, \_options.matchBrackets);

keyWinOpts.Set("directionUp"w, \_wndDirectionUp.isChecked());

keyWinOpts.Set("findText"w, \_wndFindText.getWText());

keyWinOpts.Set("replaceText"w, \_wndReplaceText.getWText());

keyWinOpts.Set("lookIn"w, \_wndLookIn.getSelection());

}

catch(Exception e)

{

return false;

}

return true;

}

bool \_ReadStateFromRegistry()

{

try

{

scope RegKey keyWinOpts = \_GetCurrentRegKey(false);

\_options.keepCase = keyWinOpts.GetDWORD("keepCase"w, \_options.keepCase) != 0;

\_options.matchCase = keyWinOpts.GetDWORD("matchCase"w, \_options.matchCase) != 0;

\_options.includePretext = keyWinOpts.GetDWORD("includePretext"w, \_options.includePretext) != 0;

\_options.matchBrackets = keyWinOpts.GetDWORD("matchBrackets"w, \_options.matchBrackets) != 0;

\_wndDirectionUp.setChecked(keyWinOpts.GetDWORD("directionUp"w, \_wndDirectionUp.isChecked()) != 0);

\_wndFindText.setText(keyWinOpts.GetString("findText"w, \_wndFindText.getWText()));

\_wndReplaceText.setText(keyWinOpts.GetString("replaceText"w, \_wndReplaceText.getWText()));

\_wndLookIn.setSelection(keyWinOpts.GetDWORD("lookIn"w, \_wndLookIn.getSelection()));

\_OptionsToDialog();

}

catch(Exception e)

{

return false;

}

return true;

}

// replaceMode -1: find last, 0: find first, 1:replace once if full match, 2+: replace all

int \_ReplaceNextInSpan(IVsTextLines buffer, IVsTextView view, int replaceMode,

int startLine, int startCol, int endLine, int endCol)

{

BSTR text;

if(buffer.GetLineText(startLine, startCol, endLine, endCol, &text) != S\_OK)

return 0;

wstring wtxt = wdetachBSTR(text);

\_options.findOnly = (replaceMode <= 0);

\_options.findMultiple = (replaceMode < 0);

wstring search = \_wndFindText.getWText();

wstring replace = \_wndReplaceText.getWText();

ReplaceRange[] ranges;

wstring ntxt = replaceTokenSequence(wtxt, startLine, startCol, search, replace, \_options, &ranges);

if(ranges.length == 0)

return 0;

if(replaceMode <= 0)

{

int idx = replaceMode < 0 ? ranges.length - 1 : 0;

if(view)

view.SetSelection(ranges[idx].startlineno, ranges[idx].startcolumn,

ranges[idx].endlineno, ranges[idx].endcolumn);

else

NavigateTo(buffer, ranges[idx].startlineno, ranges[idx].startcolumn,

ranges[idx].endlineno, ranges[idx].endcolumn);

}

else

{

if(replaceMode == 1)

{

if(ranges.length > 1)

return 0;

if(ranges[0].startlineno != startLine || ranges[0].startcolumn != startCol ||

ranges[0].endlineno != endLine || ranges[0].endcolumn != endCol)

return 0;

}

IVsCompoundAction compAct = qi\_cast!IVsCompoundAction(view);

if(compAct)

compAct.OpenCompoundAction("Replace tokens"w.ptr);

scope(exit) if(compAct)

{

compAct.CloseCompoundAction();

compAct.Release();

}

int lastReplaceLine, lastReplaceColumn;

int diffLines, diffColumns;

for(int i = 0; i < ranges.length; i++)

{

int startlineno = ranges[i].startlineno + diffLines;

int startcolumn = ranges[i].startcolumn;

int endlineno = ranges[i].endlineno + diffLines;

int endcolumn = ranges[i].endcolumn;

if(startlineno == lastReplaceLine)

startcolumn += diffColumns;

if(endlineno == lastReplaceLine)

endcolumn += diffColumns;

TextSpan changedSpan;

if(buffer.ReplaceLines(startlineno, startcolumn, endlineno, endcolumn,

ranges[i].replacementText.ptr, ranges[i].replacementText.length,

&changedSpan) != S\_OK)

return i;

diffLines += (changedSpan.iEndLine - changedSpan.iStartLine) - (endlineno - startlineno);

diffColumns = changedSpan.iEndIndex - endcolumn;

}

}

return ranges.length;

}

LRESULT \_OnFindNext()

{

bool up = \_wndDirectionUp.isChecked();

return \_DoFindNext(up);

}

LRESULT \_DoFindNext(bool up)

{

IVsTextView view;

scope(exit) release(view);

if(IVsTextLines buffer = GetCurrentTextBuffer(&view))

{

\_DialogToOptions();

scope(exit) release(buffer);

int startLine, startCol;

int endLine, endCol;

if(view)

if(!up || view.GetSelection(&startLine, &startCol, &endLine, &endCol) != S\_OK)

view.GetCaretPos (&startLine, &startCol); // caret usually at end of selection

buffer.GetLastLineIndex(&endLine, &endCol);

try

{

int found;

if(up)

{

if(startLine > 0 || startCol > 0)

found = \_ReplaceNextInSpan(buffer, view, -1, 0, 0, startLine, startCol);

if(found == 0)

found = \_ReplaceNextInSpan(buffer, view, -1, 0, 0, endLine, endCol);

}

else

{

found = \_ReplaceNextInSpan(buffer, view, 0, startLine, startCol, endLine, endCol);

if(found == 0)

if(startLine > 0 || startCol > 0)

found = \_ReplaceNextInSpan(buffer, view, 0, 0, 0, endLine, endCol);

}

if(found == 0)

{

string s = createPasteString(to!string(\_wndFindText.getWText()));

showStatusBarText("Token sequence not found: " ~ s);

}

}

catch(Exception e)

{

showStatusBarText("Token replace: " ~ e.msg);

}

}

return 0;

}

LRESULT \_OnReplace()

{

IVsTextView view;

scope(exit) release(view);

if(IVsTextLines buffer = GetCurrentTextBuffer(&view))

{

\_DialogToOptions();

scope(exit) release(buffer);

try

{

int startLine, startCol;

int endLine, endCol;

if(view && view.GetSelection(&startLine, &startCol, &endLine, &endCol) == S\_OK)

\_ReplaceNextInSpan(buffer, view, 1, startLine, startCol, endLine, endCol);

\_OnFindNext();

}

catch(Exception e)

{

showStatusBarText("Token replace: " ~ e.msg);

}

}

return 0;

}

LRESULT \_OnReplaceAll()

{

IVsTextView view;

scope(exit) release(view);

if(IVsTextLines buffer = GetCurrentTextBuffer(&view))

{

\_DialogToOptions();

scope(exit) release(buffer);

bool selOnly = (\_wndLookIn.getSelection() == 1);

int startLine, startCol;

int endLine, endCol;

if(!selOnly || !view || view.GetSelection(&startLine, &startCol, &endLine, &endCol) != S\_OK)

buffer.GetLastLineIndex(&endLine, &endCol);

try

{

int found = \_ReplaceNextInSpan(buffer, view, 2, startLine, startCol, endLine, endCol);

if(found == 0)

{

string s = createPasteString(to!string(\_wndFindText.getWText()));

showStatusBarText("Token sequence not found: " ~ s);

}

else if(found == 1)

showStatusBarText("1 token sequence replaced."w);

else

showStatusBarText(text(found, " token sequences replaced."));

}

catch(Exception e)

{

showStatusBarText("Token replace: " ~ e.msg);

}

}

return 0;

}

}

// This file is part of Visual D

//

// Visual D integrates the D programming language into Visual Studio

// Copyright (c) 2010 by Rainer Schuetze, All Rights Reserved

//

// Distributed under the Boost Software License, Version 1.0.

// See accompanying file LICENSE\_1\_0.txt or copy at <http://www.boost.org/LICENSE_1_0.txt>

module visuald.trackprojectdocument;

import visuald.windows;

import visuald.comutil;

import sdk.win32.oleauto;

import sdk.win32.objbase;

import sdk.vsi.vsshell;

import sdk.vsi.ivstrackprojectdocuments2;

import visuald.hierutil;

import visuald.hierarchy;

import visuald.chiernode;

import std.utf;

enum ProjectEventFlags

{

None = 0,

IsNestedProject = 0x1

}

// All events involving Adding, Removing, and Renaming of items in the project

// need to be announced to the IVsTrackProjectDocuments service. This service

// inturns manages broadcasting these events to interesting parties.

// For example, these events allow the Source Code Control (SCC) manager

// to coordinate SCC for the project items. These events allow the debugger to

// manage its list of breakpoints. There will be other interested parties.

//

// The class encapsulates the shell's IVsTrackProjectDocuments2 interface

// That makes it more consistent for project's rename/add/delete code.

// These methods are invoked when an project change originates internally.

// The methods just pass off to the shell methods in

// SID\_SVsTrackProjectDocuments2, which notifies other hierarchies

// that we are about to change or have changed some files.

//

class CVsTrackProjectDocuments2Helper

{

public:

this(CVsHierarchy hier)

{

mHierarchy = hier;

}

bool CanAddItem(

/\* [in] \*/ string file,

/\* [in] \*/ ProjectEventFlags flags = ProjectEventFlags.None)

{

IVsTrackProjectDocuments2 srpIVsTrackProjectDocuments2 = GetIVsTrackProjectDocuments2();

if(!srpIVsTrackProjectDocuments2)

return true;

scope(exit) release(srpIVsTrackProjectDocuments2);

IVsProject pIVsProject = cast(IVsProject) mHierarchy;

assert(pIVsProject);

VSQUERYADDFILERESULTS fSummaryResult = VSQUERYADDFILERESULTS\_AddOK;

VSQUERYADDFILEFLAGS fInputFlags = (flags & ProjectEventFlags.IsNestedProject) ? VSADDFILEFLAGS\_IsNestedProjectFile : VSADDFILEFLAGS\_NoFlags;

auto pszFile = \_toUTF16z(file);

if(SUCCEEDED(srpIVsTrackProjectDocuments2.OnQueryAddFiles(pIVsProject, 1, &pszFile,

&fInputFlags, &fSummaryResult, null)))

{

if(VSQUERYADDFILERESULTS\_AddNotOK == fSummaryResult)

return false;

}

return true;

}

void OnItemAdded(

/\* [in] \*/ CHierNode pCHierNode,

/\* [in] \*/ ProjectEventFlags flags = ProjectEventFlags.None)

{

IVsTrackProjectDocuments2 srpIVsTrackProjectDocuments2 = GetIVsTrackProjectDocuments2();

if(!srpIVsTrackProjectDocuments2)

return;

scope(exit) release(srpIVsTrackProjectDocuments2);

IVsProject pIVsProject = cast(IVsProject) mHierarchy;

assert(pIVsProject);

ScopedBSTR cbstrMkDokument;

HRESULT hr = pIVsProject.GetMkDocument(pCHierNode.GetVsItemID(), &cbstrMkDokument.bstr);

if (FAILED(hr))

return;

VSADDFILEFLAGS fInputFlags = (flags & ProjectEventFlags.IsNestedProject) ? VSADDFILEFLAGS\_IsNestedProjectFile : VSADDFILEFLAGS\_NoFlags;

wchar\*[] rgstrDocuments = [ cbstrMkDokument.bstr ];

hr = srpIVsTrackProjectDocuments2.OnAfterAddFilesEx(pIVsProject, 1, rgstrDocuments.ptr, &fInputFlags);

assert(SUCCEEDED(hr));

}

bool CanRenameItem(

/\* [in] \*/ CHierNode pCHierNode,

/\* [in] \*/ string newName,

/\* [in] \*/ ProjectEventFlags flags = ProjectEventFlags.None)

{

return true;

}

void OnItemRenamed(

/\* [in] \*/ CHierNode pCHierNode,

/\* [in] \*/ string oldName,

/\* [in] \*/ ProjectEventFlags flags = ProjectEventFlags.None)

{

}

bool CanDeleteItem(

/\* [in] \*/ CHierNode pCHierNode,

/\* [in] \*/ ProjectEventFlags flags = ProjectEventFlags.None)

{

return true;

}

void OnItemDeleted(

/\* [in] \*/ string file,

/\* [in] \*/ ProjectEventFlags flags = ProjectEventFlags.None)

{

}

protected:

IVsTrackProjectDocuments2 GetIVsTrackProjectDocuments2()

{

return queryService!(SVsTrackProjectDocuments, IVsTrackProjectDocuments2);

}

protected:

CVsHierarchy mHierarchy;

};

module visuald.vdextensions;

import sdk.port.base;

import sdk.win32.oaidl;

import sdk.win32.objbase;

import sdk.win32.oleauto;

import sdk.vsi.textmgr;

import sdk.vsi.vsshell;

import visuald.hierutil;

import visuald.dpackage;

\_\_gshared IVisualDHelper vdhelper;

interface IVisualDHelper : IUnknown

{

static const GUID iid = uuid("002a2de9-8bb6-484d-9910-7e4ad4084715");

int GetTextOptions(IVsTextView view, int\* flags, int\* tabsize, int\* indentsize);

}

IVisualDHelper createHelper()

{

if (!vdhelper)

vdhelper = VsLocalCreateInstance!IVisualDHelper (&g\_VisualDHelperCLSID, sdk.win32.wtypes.CLSCTX\_INPROC\_SERVER);

return vdhelper;

}

int vdhelper\_GetTextOptions(IVsTextView view, int\* flags, int\* tabsize, int\* indentsize)

{

if (!createHelper())

return S\_FALSE;

return vdhelper.GetTextOptions(view, flags, tabsize, indentsize);

}

// This file is part of Visual D

//

// Visual D integrates the D programming language into Visual Studio

// Copyright (c) 2010 by Rainer Schuetze, All Rights Reserved

//

// Distributed under the Boost Software License, Version 1.0.

// See accompanying file LICENSE\_1\_0.txt or copy at <http://www.boost.org/LICENSE_1_0.txt>

module visuald.vdserverclient;

import visuald.pkgutil;

import visuald.logutil;

import vdc.ivdserver;

//import vdc.semantic;

import vdc.util;

import sdk.win32.oaidl;

import sdk.win32.objbase;

import sdk.win32.oleauto;

import sdk.vsi.sdk\_shared;

import sdk.port.base;

import stdext.com;

import stdext.container;

import stdext.string;

import std.concurrency;

import std.string;

import std.conv;

import std.path;

import std.windows.charset;

import core.thread;

alias object.AssociativeArray!(string, std.concurrency.Tid) \_wa1; // fully instantiate type info for string[Tid]

alias object.AssociativeArray!(std.concurrency.Tid, string[]) \_wa2; // fully instantiate type info for string[Tid]

version(TESTMAIN) version = InProc;

debug version = DebugCmd;

// debug version = InProc;

version(InProc) import vdc.vdserver;

///////////////////////////////////////////////////////////////////////

version(DebugCmd)

{

import std.datetime;

import core.stdc.stdio : fprintf, fopen, fflush, fputc, FILE;

\_\_gshared FILE\* dbgfh;

private void dbglog(string s)

{

debug

{

version(all)

logCall("VDClient: ", s);

else

OutputDebugStringA(toMBSz("VDClient: " ~ s ~ "\n"));

}

else

{

if(!dbgfh)

dbgfh = fopen("c:/tmp/vdclient.log", "w");

SysTime now = Clock.currTime();

uint tid = sdk.win32.winbase.GetCurrentThreadId();

auto len = fprintf(dbgfh, "%02d:%02d:%02d - %04x - ",

now.hour, now.minute, now.second, tid);

fprintf(dbgfh, "%.\*s", s.length, s.ptr);

fputc('\n', dbgfh);

fflush(dbgfh);

}

}

}

///////////////////////////////////////////////////////////////////////

// can be changed through registry entry

version(debugServer)

const GUID VDServerClassFactory\_iid = uuid("002a2de9-8bb6-484d-9A02-7e4ad4084715");

else

const GUID VDServerClassFactory\_iid = uuid("002a2de9-8bb6-484d-9902-7e4ad4084715");

const GUID DParserClassFactory\_iid = uuid("002a2de9-8bb6-484d-AA05-7e4ad4084715"); // needs VDServer, not factory

\_\_gshared GUID gServerClassFactory\_iid = VDServerClassFactory\_iid;

\_\_gshared GUID IVDServer\_iid = IVDServer.iid;

\_\_gshared IClassFactory gVDClassFactory;

\_\_gshared IVDServer gVDServer;

bool startVDServer()

{

if(gVDServer)

return false;

CoInitialize(null);

version(InProc)

gVDServer = addref(newCom!VDServer);

else

{

GUID factory\_iid = IID\_IClassFactory;

HRESULT hr = CoGetClassObject(gServerClassFactory\_iid, CLSCTX\_LOCAL\_SERVER|CLSCTX\_INPROC\_SERVER, null, factory\_iid, cast(void\*\*)&gVDClassFactory);

if(FAILED(hr))

return false;

hr = gVDClassFactory.CreateInstance(null, &IVDServer\_iid, cast(void\*\*)&gVDServer);

if (FAILED(hr))

{

gVDClassFactory = release(gVDClassFactory);

return false;

}

}

version(DebugCmd) dbglog ("VDServer startet successfully");

return true;

}

bool stopVDServer()

{

if(!gVDServer)

return false;

version(DebugCmd) dbglog ("stopping VDServer");

gVDServer = release(gVDServer);

gVDClassFactory = release(gVDClassFactory);

CoUninitialize();

return true;

}

///////////////////////////////////////////////////////////////////////

struct FileCacheData

{

TextPos[] binaryIsIn;

int mParseRequestCount;

int mParseDoneCount;

}

class ServerCache

{

FileCacheData[string] mCache;

}

///////////////////////////////////////////////////////////////////////

template \_shared(T)

{

alias T \_shared;

// alias shared(T) \_shared;

}

/\*shared\*/ class Command

{

this(string cmd)

{

mCommand = cmd;

mRequest = sLastRequest++;

}

// called from clientLoop (might block due to server garbage collecting)

HRESULT exec() const

{

assert(false);

}

// polled from clientLoop (might block due to server garbage collecting)

HRESULT answer()

{

return S\_OK;

}

// called from onIdle

bool forward()

{

return true;

}

void send(Tid id)

{

.send(id, cast(size\_t) cast(void\*) this);

//                .send(id, cast(shared)this);

//                .send(id, this);

}

static uint sLastRequest;

uint mRequest;

string mCommand;

}

class ExitCommand : Command

{

this()

{

super("exit");

}

override HRESULT exec() const

{

stopVDServer();

return S\_OK;

}

}

class ClearProjectCommand : Command

{

this()

{

super("ClearProject");

}

override HRESULT exec() const

{

if(!gVDServer)

return S\_FALSE;

return gVDServer.ClearSemanticProject();

}

}

class FileCommand : Command

{

this(string cmd, string filename)

{

version(DebugCmd) cmd ~= ":" ~ baseName(filename);

super(cmd);

mFilename = filename;

}

string mFilename;

}

//////////////////////////////////////

class ConfigureProjectCommand : FileCommand

{

this(string filename, immutable(string[]) imp, immutable(string[]) stringImp,

immutable(string[]) versionids, immutable(string[]) debugids, uint flags)

{

super("ConfigureProject", filename);

mImp = imp;

mStringImp = stringImp;

mVersionids = versionids;

mDebugids = debugids;

mFlags = flags;

}

override HRESULT exec() const

{

if(!gVDServer)

return S\_FALSE;

string jimp = std.string.join(cast(string[])(mImp[]), "\n");

string jstringImp = std.string.join(cast(string[])(mStringImp[]), "\n");

string jversionids = std.string.join(cast(string[])(mVersionids[]), "\n");

string jdebugids = std.string.join(cast(string[])(mDebugids[]), "\n");

auto bfilename = allocBSTR(mFilename);

auto bimp = allocBSTR(jimp);

auto bstringImp = allocBSTR(jstringImp);

auto bversionids = allocBSTR(jversionids);

auto bdebugids = allocBSTR(jdebugids);

HRESULT hr = gVDServer.ConfigureSemanticProject(bfilename, bimp, bstringImp, bversionids, bdebugids, mFlags);

freeBSTR(bfilename);

freeBSTR(bimp);

freeBSTR(bstringImp);

freeBSTR(bversionids);

freeBSTR(bdebugids);

return hr;

}

immutable(string[]) mImp;

immutable(string[]) mStringImp;

immutable(string[]) mVersionids;

immutable(string[]) mDebugids;

uint mFlags;

}

//////////////////////////////////////

alias void delegate(uint request, string fname, string type, sdk.vsi.sdk\_shared.TextSpan span) GetTipCallBack;

class GetTipCommand : FileCommand

{

this(string filename, sdk.vsi.sdk\_shared.TextSpan span, GetTipCallBack cb)

{

super("GetTip", filename);

version(DebugCmd) mCommand ~= " {" ~ to!string(span.iStartLine) ~ "," ~ to!string(span.iStartIndex)

~ " - " ~ to!string(span.iEndLine) ~ "," ~ to!string(span.iEndIndex) ~ "}";

mSpan = span;

mCallback = cb;

}

override HRESULT exec() const

{

if(!gVDServer)

return S\_FALSE;

BSTR fname = allocBSTR(mFilename);

int iStartLine = mSpan.iStartLine + 1;

int iStartIndex = mSpan.iStartIndex;

int iEndLine = mSpan.iEndLine + 1;

int iEndIndex = mSpan.iEndIndex;

HRESULT rc = gVDServer.GetTip(fname, iStartLine, iStartIndex, iEndLine, iEndIndex);

freeBSTR(fname);

return rc;

}

override HRESULT answer()

{

if(!gVDServer)

return S\_FALSE;

BSTR btype;

int iStartLine, iStartIndex, iEndLine, iEndIndex;

HRESULT rc = gVDServer.GetTipResult(iStartLine, iStartIndex, iEndLine, iEndIndex, &btype);

if(rc != S\_OK)

return rc;

mType = detachBSTR(btype);

mSpan = sdk.vsi.sdk\_shared.TextSpan(iStartIndex, iStartLine - 1, iEndIndex, iEndLine - 1);

send(gUITid);

return S\_OK;

}

override bool forward()

{

if(mCallback)

mCallback(mRequest, mFilename, mType, mSpan);

return true;

}

GetTipCallBack mCallback;

sdk.vsi.sdk\_shared.TextSpan mSpan;

string mType;

}

//////////////////////////////////////

alias void delegate(uint request, string fname, sdk.vsi.sdk\_shared.TextSpan span) GetDefinitionCallBack;

class GetDefinitionCommand : FileCommand

{

this(string filename, sdk.vsi.sdk\_shared.TextSpan span, GetDefinitionCallBack cb)

{

super("GetDefinition", filename);

version(DebugCmd) mCommand ~= " {" ~ to!string(span.iStartLine) ~ "," ~ to!string(span.iStartIndex)

~ " - " ~ to!string(span.iEndLine) ~ "," ~ to!string(span.iEndIndex) ~ "}";

mSpan = span;

mCallback = cb;

}

override HRESULT exec() const

{

if(!gVDServer)

return S\_FALSE;

BSTR fname = allocBSTR(mFilename);

int iStartLine = mSpan.iStartLine + 1;

int iStartIndex = mSpan.iStartIndex;

int iEndLine = mSpan.iEndLine + 1;

int iEndIndex = mSpan.iEndIndex;

HRESULT rc = gVDServer.GetDefinition(fname, iStartLine, iStartIndex, iEndLine, iEndIndex);

freeBSTR(fname);

return rc;

}

override HRESULT answer()

{

if(!gVDServer)

return S\_FALSE;

BSTR fname;

int iStartLine, iStartIndex, iEndLine, iEndIndex;

HRESULT rc = gVDServer.GetDefinitionResult(iStartLine, iStartIndex, iEndLine, iEndIndex, &fname);

if(rc != S\_OK)

return rc;

mDefFile = detachBSTR(fname);

mSpan = sdk.vsi.sdk\_shared.TextSpan(iStartIndex, iStartLine - 1, iEndIndex, iEndLine - 1);

send(gUITid);

return S\_OK;

}

override bool forward()

{

if(mCallback)

mCallback(mRequest, mDefFile, mSpan);

return true;

}

GetDefinitionCallBack mCallback;

sdk.vsi.sdk\_shared.TextSpan mSpan;

string mDefFile;

}

//////////////////////////////////////

alias void delegate(uint request, string filename, string parseErrors, TextPos[] binaryIsIn) UpdateModuleCallBack;

class UpdateModuleCommand : FileCommand

{

this(string filename, wstring text, bool verbose, UpdateModuleCallBack cb)

{

super("UpdateModule", filename);

version(DebugCmd) mCommand ~= " " ~ to!string(firstLine(text));

mText = text;

mCallback = cb;

mVerbose = verbose;

}

override HRESULT exec() const

{

if(!gVDServer)

return S\_FALSE;

BSTR bfname = allocBSTR(mFilename);

BSTR btxt = allocwBSTR(mText);

HRESULT hr = gVDServer.UpdateModule(bfname, btxt, mVerbose);

freeBSTR(btxt);

freeBSTR(bfname);

return hr;

}

override HRESULT answer()

{

if(!gVDServer)

return S\_FALSE;

BSTR fname = allocBSTR(mFilename);

scope(exit) freeBSTR(fname);

BSTR errors;

if(auto hr = gVDServer.GetParseErrors(fname, &errors))

return hr;

mErrors = detachBSTR(errors);

VARIANT locs;

if(gVDServer.GetBinaryIsInLocations(fname, &locs) == S\_OK && locs.vt == VT\_ARRAY)

{

SAFEARRAY\* sa = locs.parray;

assert(SafeArrayGetDim(sa) == 1);

LONG lbound, ubound;

SafeArrayGetLBound(sa, 1, &lbound);

SafeArrayGetUBound(sa, 1, &ubound);

size\_t cnt = (ubound - lbound + 1) / 2;

mBinaryIsIn.length = cnt;

for(size\_t i = 0; i < cnt; i++)

{

LONG index = lbound + 2 \* i;

int line, col;

SafeArrayGetElement(sa, &index, &line);

mBinaryIsIn[i].line = line;

index++;

SafeArrayGetElement(sa, &index, &col);

mBinaryIsIn[i].index = col;

}

SafeArrayDestroy(sa);

}

send(gUITid);

return S\_OK;

}

override bool forward()

{

version(DebugCmd)

dbglog(to!string(mRequest) ~ " forward: " ~ mCommand ~ " " ~ ": " ~ mErrors);

if(mCallback)

mCallback(mRequest, mFilename, mErrors, cast(TextPos[])mBinaryIsIn);

return true;

}

UpdateModuleCallBack mCallback;

wstring mText;

string mErrors;

bool mVerbose;

TextPos[] mBinaryIsIn;

}

//////////////////////////////////////

alias void delegate(uint request, string filename, string tok, int line, int idx, string[] exps) GetExpansionsCallBack;

class GetExpansionsCommand : FileCommand

{

this(string filename, string tok, int line, int idx, wstring expr, GetExpansionsCallBack cb)

{

super("GetExpansions", filename);

mTok = tok;

mLine = line;

mIndex = idx;

mExpr = expr;

mCallback = cb;

}

override HRESULT exec() const

{

if(!gVDServer)

return S\_FALSE;

BSTR fname = allocBSTR(mFilename);

BSTR tok = allocBSTR(mTok);

BSTR expr = allocwBSTR(mExpr);

HRESULT rc = gVDServer.GetSemanticExpansions(fname, tok, mLine + 1, mIndex, expr);

freeBSTR(expr);

freeBSTR(tok);

freeBSTR(fname);

return rc;

}

override HRESULT answer()

{

BSTR stringList;

HRESULT rc = gVDServer.GetSemanticExpansionsResult(&stringList);

if(rc != S\_OK)

return rc;

string slist = detachBSTR(stringList);

mExpansions = /\*cast(shared(string[]))\*/ splitLines(slist);

send(gUITid);

return S\_OK;

}

override bool forward()

{

if(mCallback)

mCallback(mRequest, mFilename, mTok, mLine, mIndex, cast(string[])mExpansions);

return true;

}

GetExpansionsCallBack mCallback;

string mTok;

wstring mExpr;

int mLine;

int mIndex;

string[] mExpansions;

}

///////////////////////////////////////

alias void delegate(uint request, string filename, string tok, int line, int idx, string[] exps) GetReferencesCallBack;

class GetReferencesCommand : FileCommand

{

this(string filename, string tok, int line, int idx, wstring expr, GetReferencesCallBack cb)

{

super("GetReferences", filename);

mTok = tok;

mLine = line;

mIndex = idx;

mExpr = expr;

mCallback = cb;

}

override HRESULT exec() const

{

if(!gVDServer)

return S\_FALSE;

BSTR fname = allocBSTR(mFilename);

BSTR tok = allocBSTR(mTok);

BSTR expr = allocwBSTR(mExpr);

HRESULT rc = gVDServer.GetReferences(fname, tok, mLine + 1, mIndex, expr);

freeBSTR(expr);

freeBSTR(tok);

freeBSTR(fname);

return rc;

}

override HRESULT answer()

{

BSTR stringList;

HRESULT rc = gVDServer.GetReferencesResult(&stringList);

if(rc != S\_OK)

return rc;

string slist = detachBSTR(stringList);

mReferences = /\*cast(shared(string[]))\*/ splitLines(slist);

send(gUITid);

return S\_OK;

}

override bool forward()

{

if(mCallback)

mCallback(mRequest, mFilename, mTok, mLine, mIndex, cast(string[])mReferences);

return true;

}

GetReferencesCallBack mCallback;

string mTok;

wstring mExpr;

int mLine;

int mIndex;

string[] mReferences;

}

///////////////////////////////////////

class GetMessageCommand : Command

{

this(string message)

{

super("GetMessage");

mMessage = message;

}

override bool forward()

{

showStatusBarText(mMessage);

return true;

}

string mMessage;

}

///////////////////////////////////////////////////////////////////////

\_\_gshared Tid gUITid;

class VDServerClient

{

Tid mTid;

this()

{

}

~this()

{

shutDown();

}

void start()

{

gUITid = thisTid();

mTid = spawn(&clientLoop);

}

//////////////////////////////////////

void shutDown()

{

if(gVDServer)

{

(new \_shared!(ExitCommand)).send(mTid);

while(gVDServer)

{

Thread.sleep(dur!"msecs"(50)); // sleep for 50 milliseconds

}

}

}

//////////////////////////////////////

uint ConfigureSemanticProject(string filename, immutable(string[]) imp, immutable(string[]) stringImp,

immutable(string[]) versionids, immutable(string[]) debugids, uint flags)

{

auto cmd = new \_shared!(ConfigureProjectCommand)(filename, imp, stringImp, versionids, debugids, flags);

cmd.send(mTid);

return cmd.mRequest;

}

uint GetTip(string filename, sdk.vsi.sdk\_shared.TextSpan\* pSpan, GetTipCallBack cb)

{

auto cmd = new \_shared!(GetTipCommand)(filename, \*pSpan, cb);

cmd.send(mTid);

return cmd.mRequest;

}

uint GetDefinition(string filename, sdk.vsi.sdk\_shared.TextSpan\* pSpan, GetDefinitionCallBack cb)

{

auto cmd = new \_shared!(GetDefinitionCommand)(filename, \*pSpan, cb);

cmd.send(mTid);

return cmd.mRequest;

}

int GetSemanticExpansions(string filename, string tok, int line, int idx, wstring expr, GetExpansionsCallBack cb)

{

auto cmd = new \_shared!(GetExpansionsCommand)(filename, tok, line, idx, expr, cb);

cmd.send(mTid);

return cmd.mRequest;

}

int GetReferences(string filename, string tok, int line, int idx, wstring expr, GetReferencesCallBack cb)

{

auto cmd = new \_shared!(GetReferencesCommand)(filename, tok, line, idx, expr, cb);

cmd.send(mTid);

return cmd.mRequest;

}

uint UpdateModule(string filename, wstring text, bool verbose, UpdateModuleCallBack cb)

{

auto cmd = new \_shared!(UpdateModuleCommand)(filename, text, verbose, cb);

cmd.send(mTid);

return cmd.mRequest;

}

uint ClearSemanticProject()

{

auto cmd = new \_shared!(ClearProjectCommand);

cmd.send(mTid);

return cmd.mRequest;

}

//////////////////////////////////////

// obsolete

bool isBinaryOperator(string filename, int startLine, int startIndex, int endLine, int endIndex)

{

return false;

}

bool \_isBinaryOperator(string filename, int startLine, int startIndex, int endLine, int endIndex)

{

if(!gVDServer)

return false;

BOOL res;

BSTR fname = allocBSTR(filename);

HRESULT rc = gVDServer.IsBinaryOperator(fname, startLine, startIndex, endLine, endIndex, &res);

freeBSTR(fname);

return rc == S\_OK && res != 0;

}

bool GetParseErrors(string filename, ref string err)

{

return false;

}

bool \_GetParseErrors(string filename, ref string err)

{

if(!gVDServer)

return false;

BSTR fname = allocBSTR(filename);

scope(exit) freeBSTR(fname);

BSTR errors;

if(gVDServer.GetParseErrors(fname, &errors) != S\_OK)

return false;

err = detachBSTR(errors);

return true;

}

//////////////////////////////////////

static shared bool restartServer = false;

static void clientLoop()

{

startVDServer();

try

{

Queue!(\_shared!(Command)) toAnswer;

while(gVDServer)

{

bool changed = false;

receiveTimeout(dur!"msecs"(50),

// as of dmd 2.060, fixes of const handling expose that std.variant is not capable of working sensibly with class objects

//(shared(Command) icmd)

(size\_t icmd)

{

auto cmd = cast(Command) cast(void\*) icmd;

version(DebugCmd) dbglog(to!string(cmd.mRequest) ~ " clientLp: " ~ cmd.mCommand);

HRESULT hr = cmd.exec();

if(hr == S\_OK)

toAnswer ~= cmd;

else if((hr & 0xffff) == RPC\_S\_SERVER\_UNAVAILABLE)

restartServer = true;

changed = true;

},

(Variant var)

{

Variant var2 = var;

}

);

for(int i = 0; i < toAnswer.length && !restartServer; )

{

auto cmd = toAnswer[i];

HRESULT hr = cmd.answer();

if(hr == S\_OK)

{

toAnswer.remove(i);

changed = true;

}

else if((hr & 0xffff) == RPC\_S\_SERVER\_UNAVAILABLE)

restartServer = true;

else

i++;

}

BSTR msg;

if(gVDServer && !restartServer)

{

HRESULT hr = gVDServer.GetLastMessage(&msg);

if(hr == S\_OK)

{

string m = detachBSTR(msg);

if(m != "\_\_no\_message\_\_")

(new \_shared!(GetMessageCommand)(m)).send(gUITid);

}

else if((hr & 0xffff) == RPC\_S\_SERVER\_UNAVAILABLE)

restartServer = true;

}

version(DebugCmd) if (changed)

{

string s = " answerQ = [";

for(int i = 0; i < toAnswer.length; i++)

s ~= (i > 0 ? " " : "") ~ to!string(toAnswer[i].mRequest);

dbglog(s ~ "]");

}

if(restartServer)

{

restartServer = false;

version(DebugCmd) dbglog("\*\*\* clientLoop: restarting server \*\*\*");

stopVDServer();

startVDServer();

}

}

}

catch(Throwable e)

{

version(DebugCmd) dbglog ("clientLoop exception: " ~ e.msg);

}

stopVDServer();

}

void onIdle()

{

try

{

while(receiveTimeout(dur!"msecs"(0),

//(shared(Command) icmd)

(size\_t icmd)

{

auto cmd = cast(Command) cast(void\*) icmd;

version(DebugCmd)

if(cmd.mCommand != "GetMessage")

dbglog(to!string(cmd.mRequest) ~ " " ~ "idleLoop: " ~ cmd.mCommand);

cmd.forward();

},

(Variant var)

{

Variant var2 = var;

}

))

{

}

}

catch(Throwable e)

{

version(DebugCmd) dbglog ("clientLoop exception: " ~ e.msg);

}

}

}

// This file is part of Visual D

//

// Visual D integrates the D programming language into Visual Studio

// Copyright (c) 2010 by Rainer Schuetze, All Rights Reserved

//

// Distributed under the Boost Software License, Version 1.0.

// See accompanying file LICENSE\_1\_0.txt or copy at <http://www.boost.org/LICENSE_1_0.txt>

module visuald.viewfilter;

import visuald.windows;

import visuald.comutil;

import visuald.logutil;

import visuald.hierutil;

import visuald.fileutil;

import visuald.stringutil;

import visuald.pkgutil;

import visuald.dpackage;

import visuald.dproject;

import visuald.hierarchy;

import visuald.chiernode;

import visuald.dimagelist;

import visuald.completion;

import visuald.simpleparser;

import visuald.intellisense;

import visuald.searchsymbol;

import visuald.expansionprovider;

import visuald.dlangsvc;

import visuald.winctrl;

import visuald.tokenreplace;

import visuald.cppwizard;

import visuald.config;

import visuald.build;

import visuald.help;

import visuald.lexutil;

import vdc.lexer;

import sdk.port.vsi;

import sdk.vsi.textmgr;

import sdk.vsi.textmgr2;

import sdk.vsi.stdidcmd;

import sdk.vsi.vsshell;

import sdk.vsi.vsshell80;

import sdk.vsi.vsshell90;

import sdk.vsi.vsdbgcmd;

import sdk.vsi.vsdebugguids;

import sdk.vsi.msdbg;

import stdext.array;

import stdext.path;

import stdext.string;

import stdext.ddocmacros;

import std.string;

import std.ascii;

import std.utf;

import std.conv;

import std.algorithm;

import std.array;

import std.file;

import std.path;

///////////////////////////////////////////////////////////////////////////////

interface IVsCustomDataTip : IUnknown

{

static const GUID iid = uuid("80DD0557-F6FE-48e3-9651-398C5E7D8D78");

HRESULT DisplayDataTip();

}

// version = tip;

class ViewFilter : DisposingComObject, IVsTextViewFilter, IOleCommandTarget,

IVsTextViewEvents, IVsExpansionEvents

{

CodeWindowManager mCodeWinMgr;

IVsTextView mView;

uint mCookieTextViewEvents;

IOleCommandTarget mNextTarget;

int mLastHighlightBracesLine;

ViewCol mLastHighlightBracesCol;

version(tip)

TextTipData mTextTipData;

this(CodeWindowManager mgr, IVsTextView view)

{

mCodeWinMgr = mgr;

mView = addref(view);

mCookieTextViewEvents = Advise!(IVsTextViewEvents)(mView, this);

mView.AddCommandFilter(this, &mNextTarget);

hookWindowProc(cast(HWND) mView.GetWindowHandle());

version(tip)

mTextTipData = addref(newCom!TextTipData);

}

~this()

{

}

override void Dispose()

{

if(mView)

{

mView.RemoveCommandFilter(this);

if(mCookieTextViewEvents)

Unadvise!(IVsTextViewEvents)(mView, mCookieTextViewEvents);

mView = release(mView);

}

version(tip)

if(mTextTipData)

{

// we need to break the circular reference TextTipData<->IVsMethodTipWindow

mTextTipData.Dispose();

mTextTipData = release(mTextTipData);

}

unhookWindowProc();

mCodeWinMgr = null;

}

WNDPROC mPrevProc;

HWND mHwnd;

static ViewFilter[HWND] sHooks;

extern(Windows) static int WindowProcHook(HWND hWnd, uint uMsg, WPARAM wParam, LPARAM lParam)

{

WNDPROC proc;

ViewFilter\* pvf = hWnd in sHooks;

if (pvf)

proc = pvf.mPrevProc;

if(!proc)

proc = &DefWindowProcA;

int res = proc(hWnd,uMsg,wParam,lParam);

if(Package.GetGlobalOptions().showCoverageMargin)

if(uMsg == WM\_PAINT && pvf)

pvf.mCodeWinMgr.mSource.mColorizer.drawCoverageOverlay(hWnd, wParam, lParam, pvf.mView);

return res;

}

bool hookWindowProc(HWND hwnd)

{

if(mHwnd)

return false;

mPrevProc = cast(WNDPROC)GetWindowLongPtr(hwnd, GWL\_WNDPROC);

mHwnd = hwnd;

sHooks[mHwnd] = this;

SetWindowLongPtr(hwnd, GWL\_WNDPROC, cast(uint) &WindowProcHook);

return true;

}

bool unhookWindowProc()

{

if(!mHwnd)

return false;

SetWindowLongPtr(mHwnd, GWL\_WNDPROC, cast(uint) mPrevProc);

sHooks.remove(mHwnd);

mHwnd = null;

mPrevProc = null;

return true;

}

override HRESULT QueryInterface(in IID\* riid, void\*\* pvObject)

{

if(queryInterface!(IVsTextViewFilter) (this, riid, pvObject))

return S\_OK;

if(queryInterface!(IVsTextViewEvents) (this, riid, pvObject))

return S\_OK;

if(queryInterface!(IOleCommandTarget) (this, riid, pvObject))

return S\_OK;

if(queryInterface!(IVsExpansionEvents) (this, riid, pvObject))

return S\_OK;

return super.QueryInterface(riid, pvObject);

}

// IOleCommandTarget //////////////////////////////////////

override int QueryStatus( /\* [unique][in] \*/ in GUID \*pguidCmdGroup,

/\* [in] \*/ in uint cCmds,

/\* [out][in][size\_is] \*/ OLECMD \*prgCmds,

/\* [unique][out][in] \*/ OLECMDTEXT \*pCmdText)

{

// mixin(LogCallMix);

for (uint i = 0; i < cCmds; i++)

{

int rc = QueryCommandStatus(pguidCmdGroup, prgCmds[i].cmdID);

if(rc == E\_FAIL)

{

if(mNextTarget)

return mNextTarget.QueryStatus(pguidCmdGroup, cCmds, prgCmds, pCmdText);

return rc;

}

prgCmds[i].cmdf = cast(uint)rc;

}

return S\_OK;

}

override int Exec( /\* [unique][in] \*/ in GUID \*pguidCmdGroup,

/\* [in] \*/ in uint nCmdID,

/\* [in] \*/ in uint nCmdexecopt,

/\* [unique][in] \*/ in VARIANT \*pvaIn,

/\* [unique][out][in] \*/ VARIANT \*pvaOut)

{

if(\*pguidCmdGroup == CMDSETID\_StandardCommandSet2K && nCmdID == 1627 /\*OutputPaneCombo\*/)

return OLECMDERR\_E\_NOTSUPPORTED; // do not litter output

debug

{

bool logit = true;

if(\*pguidCmdGroup == CMDSETID\_StandardCommandSet2K)

{

switch(nCmdID)

{

case ECMD\_HANDLEIMEMESSAGE:

logit = false;

break;

default:

break;

}

}

else if(\*pguidCmdGroup == guidVSDebugCommand)

{

switch(nCmdID)

{

case cmdidOutputPaneCombo:

case cmdidProcessList:

case cmdidThreadList:

case cmdidStackFrameList:

logit = false;

break;

default:

break;

}

}

if(logit)

logCall("%s.Exec(this=%s, pguidCmdGroup=%s, nCmdId=%d: %s)",

this, cast(void\*) this, \_toLog(pguidCmdGroup), nCmdID, cmd2string(\*pguidCmdGroup, nCmdID));

}

Package.GetLanguageService().OnExec();

ushort lo = (nCmdexecopt & 0xffff);

ushort hi = (nCmdexecopt >> 16);

bool wasCompletorActive = mCodeWinMgr.mSource.IsCompletorActive();

bool gotEnterKey = false;

ExpansionProvider ep = GetExpansionProvider();

if(ep)        //if (ep.InTemplateEditingMode)

if (ep.HandlePreExec(pguidCmdGroup, nCmdID, nCmdexecopt, pvaIn, pvaOut))

return S\_OK;

if(\*pguidCmdGroup == CMDSETID\_StandardCommandSet97)

{

switch (nCmdID)

{

case cmdidPasteNextTBXCBItem:

if(PasteFromRing() == S\_OK)

return S\_OK;

break;

case cmdidGotoDefn:

return HandleGotoDef(false);

case cmdidGotoDecl:

return HandleGotoDef(true);

case cmdidFindReferences:

return HandleFindReferences();

case cmdidF1Help:

return HandleHelp();

default:

break;

}

}

if(\*pguidCmdGroup == CMDSETID\_StandardCommandSet2K)

{

switch (nCmdID)

{

case ECMD\_RETURN:

gotEnterKey = true;

break;

case ECMD\_INVOKESNIPPETFROMSHORTCUT:

return HandleSnippet();

case ECMD\_PARAMINFO:

return HandleMethodTip();

case ECMD\_FORMATSELECTION:

return ReindentLines();

case ECMD\_COMMENTBLOCK:

case ECMD\_COMMENT\_BLOCK:

return CommentLines(Source.ForceComment);

case ECMD\_UNCOMMENTBLOCK:

case ECMD\_UNCOMMENT\_BLOCK:

return CommentLines(Source.ForceUncomment);

case ECMD\_COMPLETEWORD:

case ECMD\_AUTOCOMPLETE:

if(mCodeWinMgr.mSource.IsCompletorActive())

moreCompletions();

else

initCompletion(true);

return S\_OK;

case ECMD\_SURROUNDWITH:

if (mView && ep)

//ep.DisplayExpansionBrowser(mView, "Insert Snippet", ["type1", "type2"], true, ["kind1", "kind2"], true);

ep.DisplayExpansionBrowser(mView, "Surround with", [], true, [], true);

break;

case ECMD\_INSERTSNIPPET:

if (mView && ep)

//ep.DisplayExpansionBrowser(mView, "Insert Snippet", ["type1", "type2"], true, ["kind1", "kind2"], true);

ep.DisplayExpansionBrowser(mView, "Insert Snippet", [], false, [], false);

break;

case ECMD\_COMPILE:

return CompileDoc(false, false, false, false);

case ECMD\_GOTOBRACE:

return GotoMatchingPair(false);

case ECMD\_GOTOBRACE\_EXT:

return GotoMatchingPair(true);

case ECMD\_OUTLN\_STOP\_HIDING\_ALL:

return mCodeWinMgr.mSource.StopOutlining();

case ECMD\_OUTLN\_TOGGLE\_ALL:

return mCodeWinMgr.mSource.ToggleOutlining();

default:

break;

}

}

if(g\_commandSetCLSID == \*pguidCmdGroup)

{

switch (nCmdID)

{

case CmdShowScope:

return showCurrentScope();

case CmdShowMethodTip:

return HandleMethodTip();

case CmdToggleComment:

return CommentLines(Source.AutoComment);

case CmdConvSelection:

return ConvertSelection();

case CmdCompileAndRun:

return CompileDoc(true, true, false, false);

case CmdCompileAndDbg:

return CompileDoc(true, false, true, false);

case CmdCompileAndAsm:

return CompileDoc(false, false, false, true);

case CmdCollapseUnittest:

return mCodeWinMgr.mSource.CollapseDisabled(true, false);

case CmdCollapseDisabled:

return mCodeWinMgr.mSource.CollapseDisabled(false, true);

default:

break;

}

}

/+

switch (lo)

{

case OLECMDEXECOPT.OLECMDEXECOPT\_SHOWHELP:

if((nCmdexecopt >> 16) == VsMenus.VSCmdOptQueryParameterList) {

return QueryParameterList(ref guidCmdGroup, nCmdId, nCmdexecopt, pvaIn, pvaOut);

}

break;

default:

// On every command, update the tip window if it's active.

if(this.textTipData != null && this.textTipData.IsActive())

textTipData.CheckCaretPosition(this.textView);

int rc = 0;

try {

rc = ExecCommand(ref guidCmdGroup, nCmdId, nCmdexecopt, pvaIn, pvaOut);

} catch (COMException e) {

int hr = e.ErrorCode;

// We silently fail on the following errors because the user has

// most likely already been prompted with things like source control checkout

// dialogs and so forth.

if(hr != (int)TextBufferErrors.BUFFER\_E\_LOCKED &&

hr != (int)TextBufferErrors.BUFFER\_E\_READONLY &&

hr != (int)TextBufferErrors.BUFFER\_E\_READONLY\_REGION &&

hr != (int)TextBufferErrors.BUFFER\_E\_SCC\_READONLY) {

throw;

}

}

return rc;

}

return OLECMDERR\_E\_NOTSUPPORTED;

+/

int rc = mNextTarget.Exec(pguidCmdGroup, nCmdID, nCmdexecopt, pvaIn, pvaOut);

if (ep)

if (ep.HandlePostExec(pguidCmdGroup, nCmdID, nCmdexecopt, gotEnterKey, pvaIn, pvaOut))

return rc;

if(\*pguidCmdGroup == CMDSETID\_StandardCommandSet97)

{

switch (nCmdID)

{

case cmdidPasteNextTBXCBItem:

case cmdidPaste:

ReindentPastedLines();

break;

default:

break;

}

}

if(\*pguidCmdGroup == CMDSETID\_StandardCommandSet2K)

{

switch (nCmdID)

{

case ECMD\_RETURN:

if(!wasCompletorActive)

HandleSmartIndent('\n');

break;

case ECMD\_LEFT:

case ECMD\_WORDPREV:

case ECMD\_RIGHT:

case ECMD\_WORDNEXT:

stopCompletions();

goto case ECMD\_UP;

case ECMD\_BACKSPACE:

if(mCodeWinMgr.mSource.IsCompletorActive())

initCompletion(false);

goto case;

case ECMD\_UP:

case ECMD\_DOWN:

if(mCodeWinMgr.mSource.IsMethodTipActive())

HandleMethodTip();

break;

case ECMD\_TYPECHAR:

dchar ch = pvaIn.uiVal;

if(ch == '.' && Package.GetGlobalOptions().expandTrigger >= 1 && Package.GetGlobalOptions().expandFromSemantics)

initCompletion(false);

else if(mCodeWinMgr.mSource.IsCompletorActive() || Package.GetGlobalOptions().expandTrigger >= 2)

{

if(dLex.isIdentifierChar(ch))

initCompletion(false);

else

stopCompletions();

}

if(ch == '{' || ch == '}' || ch == '[' || ch == ']' ||

ch == 'n' || ch == 't' || ch == 'y') // last characters of "in", "out" and "body"

HandleSmartIndent(ch);

if(mCodeWinMgr.mSource.IsMethodTipActive())

{

if(ch == ',' || ch == ')')

HandleMethodTip();

}

else if(ch == '(')

{

LANGPREFERENCES langPrefs;

if(GetUserPreferences(&langPrefs, null) == S\_OK && langPrefs.fAutoListParams)

\_HandleMethodTip(false);

}

break;

default:

if(nCmdID < ECMD\_FINAL && nCmdID != ECMD\_HANDLEIMEMESSAGE)

stopCompletions();

break;

}

}

// delayed into idle: HighlightMatchingPairs();

return rc;

}

//////////////////////////////

HRESULT CompileDoc(bool rdmd, bool run, bool dbg, bool disasm)

{

IVsUIShellOpenDocument pIVsUIShellOpenDocument = queryService!(IVsUIShellOpenDocument);

if(!pIVsUIShellOpenDocument)

return returnError(E\_FAIL);

scope(exit) release(pIVsUIShellOpenDocument);

string fname = mCodeWinMgr.mSource.GetFileName();

wchar\* wfname = \_toUTF16z(fname);

string addopt;

Config cfg;

CFileNode pFile;

IVsUIHierarchy pUIH;

uint itemid;

IServiceProvider pSP;

VSDOCINPROJECT docInProj;

if(pIVsUIShellOpenDocument.IsDocumentInAProject(wfname, &pUIH, &itemid, &pSP, &docInProj) != S\_OK)

return S\_OK;

scope(exit) release(pSP);

scope(exit) release(pUIH);

if(!pUIH)

return returnError(E\_FAIL);

Project proj = qi\_cast!Project(pUIH);

scope(exit) release(proj);

BSTR bstrSelText;

string selText;

if(mView.GetSelectedText(&bstrSelText) == S\_OK && !disasm)

selText = detachBSTR(bstrSelText);

if(!proj)

{

// not in Visual D project, but in workspace project

ProjectFactory factory = newCom!ProjectFactory(Package.s\_instance);

string filename = normalizeDir(tempDir()) ~ "\_\_compile\_\_.vdproj";

proj = newCom!Project(factory, "\_\_compile\_\_", filename, "Debug", "Win32").addref();

pFile = newCom!CFileNode(fname);

proj.GetProjectNode().Add(pFile);

IVsCfgProvider pCfgProvider;

IVsCfg icfg;

scope(exit) release(pCfgProvider);

scope(exit) release(icfg);

if(proj.GetCfgProvider(&pCfgProvider) == S\_OK)

if(pCfgProvider.GetCfgs(1, &icfg, null, null) == S\_OK)

cfg = qi\_cast!Config(icfg);

if(cfg)

{

cfg.GetProjectOptions().outdir = normalizeDir(tempDir()) ~ "\_\_vdcompile";

cfg.GetProjectOptions().release = false;

}

string modname = getModuleDeclarationName(fname);

if(modname.length)

{

// add the path that seems to be the root of the package

string ipath;

while(findSkip(modname, "."))

ipath ~= "/..";

if(ipath.length)

addopt ~= " -I" ~ normalizeDir(dirName(fname)) ~ ipath[1..$];

}

}

else

{

CHierNode pNode = proj.VSITEMID2Node(itemid);

if(!pNode)

return returnError(E\_INVALIDARG);

pFile = cast(CFileNode) pNode;

if(!pFile)

return S\_OK;

auto solutionBuildManager = queryService!(IVsSolutionBuildManager)();

scope(exit) release(solutionBuildManager);

IVsProjectCfg activeCfg;

scope(exit) release(activeCfg);

if(solutionBuildManager)

if(solutionBuildManager.FindActiveProjectCfg(null, null, proj, &activeCfg) == S\_OK)

cfg = qi\_cast!Config(activeCfg);

}

if(!cfg || !pFile)

return S\_OK;

if(pFile.SaveDoc(SLNSAVEOPT\_SaveIfDirty) != S\_OK)

return returnError(E\_FAIL);

mCodeWinMgr.mSource.OnBufferSave(null); // save current modification position

auto symdebug = cfg.GetProjectOptions().symdebug;

scope(exit)

{

cfg.GetProjectOptions().symdebug = symdebug;

release(cfg);

}

if (disasm && symdebug == 0) // ensure debug info is enabled

cfg.GetProjectOptions().symdebug = 3;

string stool = cfg.GetStaticCompileTool(pFile, cfg.getCfgName());

if(stool == "DMD")

stool = "DMDsingle";

if(stool == "DMDsingle" && rdmd)

{

stool = "RDMD";

if(selText.length)

{

string[] lines = splitLines(selText);

foreach(ln; lines)

{

string line = strip(detab(ln));

if(line.length)

addopt ~= " \"--eval=" ~ replace(line, "\"", "\\\\\\\"") ~ "\"";

}

}

}

if(stool == "RDMD" && run)

addopt = " --build-only " ~ Package.GetGlobalOptions().compileAndRunOpts ~ addopt;

else if(stool == "RDMD" && dbg)

addopt = " --build-only " ~ Package.GetGlobalOptions().compileAndDbgOpts ~ addopt;

string cmd = cfg.GetCompileCommand(pFile, !dbg && !run && !disasm, stool, addopt);

if(cmd.length)

{

cmd ~= "if not errorlevel 1 echo Compilation successful.\n";

string workdir = cfg.GetProjectDir();

string outfile = cfg.GetOutputFile(pFile, stool);

outfile = makeFilenameAbsolute(outfile, workdir);

string cmdfile = outfile ~ ".syntax";

removeCachedFileTime(outfile);

if(disasm)

{

string asmfile = outfile ~ ".asm";

string linfile = outfile ~ ".lines";

cmd ~= "if errorlevel 1 exit %ERRORLEVEL% /B\n";

cmd ~= "echo Dumping disassembly\n";

cmd ~= cfg.GetDisasmCommand(outfile, asmfile) ~ "\n";

cmd ~= "if errorlevel 1 exit %ERRORLEVEL% /B\n";

cmd ~= "echo Dumping line numbers\n";

cmd ~= "\"" ~ Package.GetGlobalOptions().VisualDInstallDir ~ "cv2pdb\\dumplines.exe\" " ~ quoteFilename(outfile)

~ " > " ~ quoteFilename(linfile) ~ "\n";

}

if(run)

{

cmd ~= "if errorlevel 1 exit %ERRORLEVEL% /B\n";

cmd ~= quoteFilename(outfile) ~ "\n";

cmd ~= "echo Execution result code: %ERRORLEVEL%\n";

}

auto pane = getVisualDOutputPane();

scope(exit) release(pane);

clearOutputPane();

if(pane)

pane.Activate();

HRESULT hr = RunCustomBuildBatchFile(outfile, cmdfile, cmd, pane, cfg.getBuilder());

if(run)

Package.GetGlobalOptions().addExecutionPath(workdir, null);

if(hr == S\_OK)

{

if(dbg)

cfg.\_DebugLaunch(outfile, dirName(fname), null, Package.GetGlobalOptions().compileAndDbgEngine);

if(disasm)

mCodeWinMgr.mSource.setDisasmFiles(outfile ~ ".asm", outfile ~ ".lines");

}

}

return S\_OK;

}

//////////////////////////////

void initCompletion(bool autoInsert)

{

CompletionSet cs = mCodeWinMgr.mSource.GetCompletionSet();

Declarations decl = new Declarations;

decl.StartExpansions(mView, mCodeWinMgr.mSource, autoInsert);

}

void moreCompletions()

{

CompletionSet cs = mCodeWinMgr.mSource.GetCompletionSet();

Declarations decl = cs.mDecls;

decl.MoreExpansions(mView, mCodeWinMgr.mSource);

}

void stopCompletions()

{

if(CompletionSet cs = mCodeWinMgr.mSource.GetCompletionSet())

if(Declarations decl = cs.mDecls)

decl.StopExpansions();

mCodeWinMgr.mSource.DismissCompletor();

}

int QueryCommandStatus(in GUID \*guidCmdGroup, uint cmdID)

{

if(\*guidCmdGroup == CMDSETID\_StandardCommandSet97)

{

switch (cmdID)

{

case cmdidPasteNextTBXCBItem:

return OLECMDF\_SUPPORTED | OLECMDF\_ENABLED;

case cmdidGotoDefn:

case cmdidGotoDecl:

case cmdidFindReferences:

//case VsCommands.GotoDecl:

//case VsCommands.GotoRef:

return OLECMDF\_SUPPORTED | OLECMDF\_ENABLED;

default:

break;

}

}

if(\*guidCmdGroup == CMDSETID\_StandardCommandSet2K)

{

switch (cmdID)

{

case ECMD\_PARAMINFO:

case ECMD\_FORMATSELECTION:

case ECMD\_COMMENTBLOCK:

case ECMD\_COMMENT\_BLOCK:

case ECMD\_UNCOMMENTBLOCK:

case ECMD\_UNCOMMENT\_BLOCK:

case ECMD\_COMPLETEWORD:

case ECMD\_INSERTSNIPPET:

case ECMD\_INVOKESNIPPETFROMSHORTCUT:

case ECMD\_SURROUNDWITH:

case ECMD\_AUTOCOMPLETE:

case ECMD\_GOTOBRACE:

case ECMD\_GOTOBRACE\_EXT:

case ECMD\_OUTLN\_STOP\_HIDING\_ALL:

case ECMD\_OUTLN\_TOGGLE\_ALL:

case ECMD\_COMPILE:

return OLECMDF\_SUPPORTED | OLECMDF\_ENABLED;

default:

break;

}

}

if(g\_commandSetCLSID == \*guidCmdGroup)

{

switch (cmdID)

{

case CmdShowScope:

case CmdShowMethodTip:

case CmdToggleComment:

case CmdConvSelection:

case CmdCompileAndRun:

case CmdCompileAndDbg:

case CmdCompileAndAsm:

case CmdCollapseUnittest:

case CmdCollapseDisabled:

return OLECMDF\_SUPPORTED | OLECMDF\_ENABLED;

default:

break;

}

}

return E\_FAIL;

}

int HighlightComment(wstring txt, int line, ref ViewCol idx, out int otherLine, out int otherIndex)

{

int iState, tokidx;

uint pos;

size\_t idxpos = idx;

if(Lexer.isStartingComment(txt, idxpos))

{

idx = cast(ViewCol) idxpos;

tokidx = mCodeWinMgr.mSource.FindLineToken(line, idx, iState, pos);

if(pos == idx)

{

int startState = iState;

if(dLex.scan(iState, txt, pos) == TokenCat.Comment)

{

//if(iState == Lexer.toState(Lexer.State.kNestedComment, 1, 0) ||

if(iState == Lexer.State.kWhite)

{

// terminated on same line

otherLine = line;

otherIndex = pos - 2; //assume 2 character comment extro

return S\_OK;

}

if(Lexer.isCommentState(Lexer.scanState(iState)))

{

if(mCodeWinMgr.mSource.FindEndOfComment(startState, iState, line, pos))

{

otherLine = line;

otherIndex = pos - 2; //assume 2 character comment extro

return S\_OK;

}

}

}

}

}

if(Lexer.isEndingComment(txt, idxpos))

{

idx = cast(ViewCol) idxpos;

tokidx = mCodeWinMgr.mSource.FindLineToken(line, idx, iState, pos);

if(tokidx >= 0)

{

int startState = iState;

uint startpos = pos;

if(dLex.scan(iState, txt, pos) == TokenCat.Comment)

{

if(startState == iState ||

mCodeWinMgr.mSource.FindStartOfComment(startState, line, startpos))

{

otherLine = line;

otherIndex = startpos;

return S\_OK;

}

}

/+

int prevpos = pos;

int prevline = line;

Lexer.scan(iState, txt, pos);

if(pos == idx + 2 && iState == Lexer.State.kWhite)

{

while(line > 0)

{

TokenInfo[] lineInfo = mCodeWinMgr.mSource.GetLineInfo(line);

if(tokidx < 0)

tokidx = lineInfo.length - 1;

while(tokidx >= 0)

{

if(lineInfo[tokidx].type != TokenCat.Comment)

{

otherLine = prevline;

otherIndex = prevpos;

return S\_OK;

}

prevpos = lineInfo[tokidx].StartIndex;

prevline = line;

tokidx--;

}

line--;

}

}

+/

}

}

return S\_FALSE;

}

int HighlightString(wstring txt, int line, ref ViewCol idx, out int otherLine, out int otherIndex)

{

int iState;

uint pos;

auto src = mCodeWinMgr.mSource;

int tokidx = src.FindLineToken(line, idx, iState, pos);

if(tokidx < 0)

return S\_FALSE;

uint startPos = pos;

int startState = iState;

int type = dLex.scan(iState, txt, pos);

if(type == TokenCat.String)

{

Lexer.State sstate;

sstate = Lexer.scanState(startState);

if(idx == startPos && !Lexer.isStringState(sstate))

{

if(src.FindEndOfString(startState, iState, line, pos))

{

otherLine = line;

otherIndex = pos - 1;

return S\_OK;

}

return S\_FALSE;

}

sstate = Lexer.scanState(iState);

if(idx == pos - 1 && !Lexer.isStringState(sstate))

{

if(src.FindStartOfString(startState, line, startPos))

{

otherLine = line;

otherIndex = startPos;

return S\_OK;

}

}

}

return S\_FALSE;

}

int HighlightMatchingPairs()

{

int line, otherLine;

ViewCol idx, otherIndex;

int highlightLen;

bool checkMismatch;

if(int rc = mView.GetCaretPos(&line, &idx))

return rc;

if(FindMatchingPairs(line, idx, otherLine, otherIndex, highlightLen, checkMismatch) != S\_OK)

return S\_OK;

TextSpan[2] spans;

spans[0].iStartLine = line;

spans[0].iStartIndex = idx;

spans[0].iEndLine = line;

spans[0].iEndIndex = idx + highlightLen;

spans[1].iStartLine = otherLine;

spans[1].iStartIndex = otherIndex;

spans[1].iEndLine = otherLine;

spans[1].iEndIndex = otherIndex + highlightLen;

// HIGHLIGHTMATCHINGBRACEFLAGS.USERECTANGLEBRACES

HRESULT hr = mView.HighlightMatchingBrace(0, 2, spans.ptr);

if(highlightLen == 1 && checkMismatch)

{

wstring txt = mCodeWinMgr.mSource.GetText(line, idx, line, idx + 1);

wstring otxt = mCodeWinMgr.mSource.GetText(otherLine, otherIndex, otherLine, otherIndex + 1);

if(!otxt.length || !Lexer.isBracketPair(txt[0], otxt[0]))

showStatusBarText("mismatched bracket " ~ otxt);

}

return hr;

}

int FindMatchingPairs(int line, ref ViewCol idx, out int otherLine, out ViewCol otherIndex,

                 out int highlightLen, out bool checkMismatch)

{

wstring txt = mCodeWinMgr.mSource.GetText(line, 0, line, -1);

if(txt.length <= idx)

return S\_FALSE;

highlightLen = 1;

checkMismatch = true;

if(HighlightComment(txt, line, idx, otherLine, otherIndex) == S\_OK)

highlightLen = 2;

else if(HighlightString(txt, line, idx, otherLine, otherIndex) == S\_OK)

checkMismatch = false;

else if(!Lexer.isOpeningBracket(txt[idx]) &&

!Lexer.isClosingBracket(txt[idx]))

return S\_FALSE;

else if(!FindMatchingBrace(line, idx, otherLine, otherIndex))

{

// showStatusBarText("no matching bracket found"w);

return S\_FALSE;

}

return S\_OK;

}

bool FindMatchingBrace(int line, int idx, out int otherLine, out int otherIndex)

{

int iState;

uint pos;

int tok = mCodeWinMgr.mSource.FindLineToken(line, idx, iState, pos);

if(tok < 0)

return false;

wstring text = mCodeWinMgr.mSource.GetText(line, 0, line, -1);

uint ppos = pos;

int toktype = dLex.scan(iState, text, pos);

if(toktype != TokenCat.Operator)

return false;

if(Lexer.isOpeningBracket(text[ppos]))

return mCodeWinMgr.mSource.FindClosingBracketForward(line, iState, pos, otherLine, otherIndex);

else if(Lexer.isClosingBracket(text[ppos]))

return mCodeWinMgr.mSource.FindOpeningBracketBackward(line, tok, otherLine, otherIndex);

return false;

}

int FindClosingMatchingPairs(out int line, out ViewCol idx, out int otherLine, out ViewCol otherIndex,

                                 out int highlightLen, out bool checkMismatch)

{

if(int rc = mView.GetCaretPos(&line, &idx))

return rc;

int caretLine = line;

int caretIndex = idx;

while(line >= 0)

{

wstring text = mCodeWinMgr.mSource.GetText(line, 0, line, -1);

if(idx < 0)

idx = text.length;

while(--idx >= 0)

{

if(Lexer.isOpeningBracket(text[idx]) ||

text[idx] == '\"' || text[idx] == '`' || text[idx] == '/')

{

if(FindMatchingPairs(line, idx, otherLine, otherIndex, highlightLen, checkMismatch) == S\_OK)

if(otherLine > caretLine ||

(otherLine == caretLine && otherIndex > caretIndex))

return S\_OK;

}

}

line--;

}

return S\_FALSE;

}

int GotoMatchingPair(bool select)

{

int line, otherLine;

ViewCol idx, otherIndex;

int highlightLen;

bool checkMismatch;

if(mView.GetCaretPos(&line, &idx) != S\_OK)

return S\_FALSE;

if(FindMatchingPairs(line, idx, otherLine, otherIndex, highlightLen, checkMismatch) != S\_OK)

if(FindClosingMatchingPairs(line, idx, otherLine, otherIndex, highlightLen, checkMismatch) != S\_OK)

return S\_OK;

mView.SetCaretPos(otherLine, otherIndex);

TextSpan span;

span.iStartLine = otherLine;

span.iStartIndex = otherIndex;

span.iEndLine = otherLine;

span.iEndIndex = otherIndex + highlightLen;

mView.EnsureSpanVisible(span);

if(select)

mView.SetSelection (line, idx, otherLine, otherIndex + highlightLen);

return S\_OK;

}

//////////////////////////////

wstring GetWordAtCaret()

{

int line, idx;

if(mView.GetCaretPos(&line, &idx) != S\_OK)

return "";

int startIdx, endIdx;

if(!mCodeWinMgr.mSource.GetWordExtent(line, idx, WORDEXT\_CURRENT, startIdx, endIdx))

return "";

return mCodeWinMgr.mSource.GetText(line, startIdx, line, endIdx);

}

ExpansionProvider GetExpansionProvider()

{

return mCodeWinMgr.mSource.GetExpansionProvider();

}

int HandleSnippet()

{

int line, idx;

if(mView.GetCaretPos(&line, &idx) != S\_OK)

return S\_FALSE;

int startIdx, endIdx;

if(!mCodeWinMgr.mSource.GetWordExtent(line, idx, WORDEXT\_CURRENT, startIdx, endIdx))

return S\_FALSE;

wstring shortcut = mCodeWinMgr.mSource.GetText(line, startIdx, line, endIdx);

TextSpan ts = TextSpan(startIdx, line, endIdx, line);

string title, path;

ExpansionProvider ep = GetExpansionProvider();

return ep.InvokeExpansionByShortcut(mView, shortcut, ts, true, title, path);

}

//////////////////////////////////////////////////////////////

int showCurrentScope()

{

TextSpan span;

if(mView.GetCaretPos(&span.iStartLine, &span.iStartIndex) != S\_OK)

return S\_FALSE;

int line = span.iStartLine;

int idx = span.iStartIndex;

int iState;

uint pos;

int tok = mCodeWinMgr.mSource.FindLineToken(line, idx, iState, pos);

wstring curScope;

int otherLine, otherIndex;

Source src = mCodeWinMgr.mSource;

while(src.FindOpeningBracketBackward(line, tok, otherLine, otherIndex))

{

tok = mCodeWinMgr.mSource.FindLineToken(line, otherIndex, iState, pos);

wstring bracket = src.GetText(otherLine, otherIndex, otherLine, otherIndex + 1);

if(bracket == "{"w)

{

wstring fn;

src.findStatementStart(otherLine, otherIndex, fn);

wstring name = src.getScopeIdentifer(otherLine, otherIndex, fn);

if(name.length && name != "{")

{

if(curScope.length)

curScope = "." ~ curScope;

curScope = name ~ curScope;

}

}

line = otherLine;

}

if(curScope.length)

showStatusBarText("Scope: " ~ curScope);

else

showStatusBarText("Scope: at module scope"w);

return S\_OK;

}

//////////////////////////////////////////////////////////////

int ConvertSelection()

{

if(convertSelection(mView))

return S\_OK;

return S\_FALSE;

}

//////////////////////////////////////////////////////////////

int HandleSmartIndent(dchar ch)

{

LANGPREFERENCES langPrefs;

if(int rc = GetUserPreferences(&langPrefs, mView))

return rc;

if(langPrefs.IndentStyle != vsIndentStyleSmart)

return S\_FALSE;

int line, idx, len;

if(int rc = mView.GetCaretPos(&line, &idx))

return rc;

if(ch != '\n')

idx--;

else if(mCodeWinMgr.mSource.mBuffer.GetLengthOfLine(line, &len) == S\_OK && len > 0)

return ReindentLines();

wstring linetxt = mCodeWinMgr.mSource.GetText(line, 0, line, -1);

int p, orgn = countVisualSpaces(linetxt, langPrefs.uTabSize, &p);

wstring trimmed;

if(std.ascii.isAlpha(ch) && ((trimmed = strip(linetxt)) == "in" || trimmed == "out" || trimmed == "body"))

return ReindentLines();

if(idx > p || (ch != '\n' && linetxt[p] != ch))

return S\_FALSE; // do nothing if not at beginning of line

Source.CacheLineIndentInfo cacheInfo;

int n = mCodeWinMgr.mSource.CalcLineIndent(line, ch, &langPrefs, cacheInfo);

if(n < 0 || n == orgn)

return S\_OK;

if(ch == '\n')

return mView.SetCaretPos(line, n);

else

return mCodeWinMgr.mSource.doReplaceLineIndent(line, p, n, &langPrefs);

}

int ReindentLines()

{

int iStartLine, iStartIndex, iEndLine, iEndIndex;

int hr = GetSelectionForward(mView, &iStartLine, &iStartIndex, &iEndLine, &iEndIndex);

if(FAILED(hr)) // S\_FALSE if no selection, but caret-coordinates returned

return hr;

return ReindentLines(iStartLine, iEndLine);

}

int ReindentLines(int iStartLine, int iEndLine)

{

if(iEndLine < iStartLine)

std.algorithm.swap(iStartLine, iEndLine);

IVsCompoundAction compAct = qi\_cast!IVsCompoundAction(mView);

if(compAct)

compAct.OpenCompoundAction("ReindentLines"w.ptr);

int hr = mCodeWinMgr.mSource.ReindentLines(mView, iStartLine, iEndLine);

if(compAct)

{

compAct.CloseCompoundAction();

compAct.Release();

}

return hr;

}

int ReindentPastedLines()

{

if(Package.GetGlobalOptions().pasteIndent)

with(mCodeWinMgr.mSource.mLastTextLineChange)

if(iStartLine != iNewEndLine)

return ReindentLines(iStartLine, iNewEndLine);

return S\_OK;

}

//////////////////////////////////////////////////////////////

int CommentLines(int commentMode)

{

int iStartLine, iStartIndex, iEndLine, iEndIndex;

int hr = GetSelectionForward(mView, &iStartLine, &iStartIndex, &iEndLine, &iEndIndex);

if(FAILED(hr)) // S\_FALSE if no selection, but caret-coordinates returned

return hr;

if(iEndIndex == 0 && iEndLine > iStartLine)

iEndLine--;

IVsCompoundAction compAct = qi\_cast!IVsCompoundAction(mView);

if(compAct)

compAct.OpenCompoundAction("CommentLines"w.ptr);

hr = mCodeWinMgr.mSource.CommentLines(mView, iStartLine, iEndLine, commentMode);

if(compAct)

{

compAct.CloseCompoundAction();

compAct.Release();

}

return hr;

}

//////////////////////////////////////////////////////////////

int PasteFromRing()

{

if(auto svc = queryService!(IVsToolbox, IVsToolboxClipboardCycler))

{

scope(exit) release(svc);

wstring[] entries;

int[] entryIndex;

int cntEntries = 0;

svc.BeginCycle();

IVsToolboxUser tbuser = qi\_cast!IVsToolboxUser(mView);

scope(exit) release(tbuser);

BOOL itemsAvailable;

if(svc.AreDataObjectsAvailable(tbuser, &itemsAvailable) == S\_OK && itemsAvailable)

{

IDataObject firstDataObject;

IDataObject pDataObject;

while(entries.length < 30 &&

svc.GetAndSelectNextDataObject(tbuser, &pDataObject) == S\_OK)

{

scope(exit) release(pDataObject);

if(pDataObject is firstDataObject)

break;

if(!firstDataObject)

firstDataObject = addref(pDataObject);

FORMATETC fmt;

fmt.cfFormat = CF\_UNICODETEXT;

fmt.ptd = null;

fmt.dwAspect = DVASPECT\_CONTENT;

fmt.lindex = -1;

fmt.tymed = TYMED\_HGLOBAL;

STGMEDIUM medium;

if(pDataObject.GetData(&fmt, &medium) == S\_OK)

{

if(medium.tymed == TYMED\_HGLOBAL)

{

wstring s = UtilGetStringFromHGLOBAL(medium.hGlobal);

.GlobalFree(medium.hGlobal);

s = createPasteString(s);

if(!contains(entries, s))

{

entries ~= s;

entryIndex ~= cntEntries;

}

}

}

cntEntries++;

}

release(firstDataObject);

if(entries.length > 0)

{

TextSpan span;

if(mView.GetCaretPos (&span.iStartLine, &span.iStartIndex) == S\_OK)

{

span.iEndLine = span.iStartLine;

span.iEndIndex = span.iStartIndex;

mView.EnsureSpanVisible(span);

POINT pt;

if(mView.GetPointOfLineColumn (span.iStartLine, span.iStartIndex, &pt) == S\_OK)

{

int height;

mView.GetLineHeight (&height);

pt.y += height;

HWND hwnd = cast(HWND) mView.GetWindowHandle();

ClientToScreen(hwnd, &pt);

for(int k = 0; k < 10 && k < entries.length; k++)

entries[k] = entries[k] ~ "\t(&" ~ cast(wchar)('0' + ((k + 1) % 10)) ~ ")";

int sel = PopupContextMenu(hwnd, pt, entries);

if(sel >= 0 && sel < entryIndex.length)

{

int cnt = entryIndex[sel];

svc.BeginCycle();

for(int i = 0; i <= cnt; i++)

{

if(svc.GetAndSelectNextDataObject(tbuser, &pDataObject) == S\_OK)

release(pDataObject);

}

return E\_NOTIMPL; // forward to VS for insert

}

}

}

}

return S\_OK; // do not pass to VS, insert cancelled

}

}

return E\_NOTIMPL; // forward to VS for insert

}

//////////////////////////////////////////////////////////////

int RemoveUnittests()

{

int endLine, endCol;

mCodeWinMgr.mSource.GetLastLineIndex(endLine, endCol);

wstring wtxt = mCodeWinMgr.mSource.GetText(0, 0, endLine, endCol);

ReplaceOptions opt;

version(none)

{

string txt = to!string(wtxt);

string rtxt = replaceTokenSequence(txt, "unittest { $any }", "", opt, null);

if(txt == rtxt)

return S\_OK;

wstring wrtxt = to!wstring(rtxt);

}

else

wstring wrtxt = replaceTokenSequence(wtxt, 1, 0, "unittest { $any }", "", opt, null);

TextSpan changedSpan;

return mCodeWinMgr.mSource.mBuffer.ReplaceLines(0, 0, endLine, endCol, wrtxt.ptr, wrtxt.length, &changedSpan);

}

//////////////////////////////////////////////////////////////

int HandleGotoDef(bool decl)

{

int line, idx;

if(mView.GetCaretPos(&line, &idx) != S\_OK)

return S\_FALSE;

string file = mCodeWinMgr.mSource.GetFileName();

wstring impw = mCodeWinMgr.mSource.GetImportModule(line, idx, false);

if(impw.length)

{

string imp = to!string(impw);

imp = replace(imp, ".", "\\") ~ ".d";

HRESULT hr = OpenFileInSolution(imp, -1, 0, file, false); // also searches import paths

if(hr != S\_OK)

{

imp ~= "i";

hr = OpenFileInSolution(imp, -1, 0, file, false);

if(hr != S\_OK)

{

imp = imp[0 .. $-3] ~ "[\\package.d](file:///\\package.d)";

hr = OpenFileInSolution(imp, -1, 0, file, false);

}

}

return hr;

}

if(Package.GetGlobalOptions().semanticGotoDef)

{

TextSpan span;

span.iStartLine = span.iEndLine = line;

span.iStartIndex = span.iEndIndex = idx;

if (!mCodeWinMgr.mSource.GetTipSpan(&span))

return S\_FALSE;

mLastGotoDecl = decl;

mLastGotoDef = to!string(mCodeWinMgr.mSource.GetText(span.iStartLine, span.iStartIndex, span.iEndLine, span.iEndIndex));

Package.GetLanguageService().GetDefinition(mCodeWinMgr.mSource, &span, &GotoDefinitionCallBack);

return S\_FALSE;

}

else

{

string word = toUTF8(GetWordAtCaret());

if(word.length <= 0)

return S\_FALSE;

return GotoDefinitionJSON(file, word);

}

}

version(all)

HRESULT GotoDefinitionCPP(string word)

{

if(auto objmgr = queryService!(IVsObjectManager))

{

scope(exit) release(objmgr);

if(auto objmgr2 = qi\_cast!IVsObjectManager2(objmgr))

{

scope(exit) release(objmgr2);

IVsEnumLibraries2 enumLibs;

if(objmgr2.EnumLibraries(&enumLibs) == S\_OK)

{

VSOBSEARCHCRITERIA2 searchOpts;

searchOpts.szName = \_toUTF16z(word);

searchOpts.eSrchType = SO\_ENTIREWORD;

searchOpts.grfOptions = VSOBSO\_CASESENSITIVE;

scope(exit) release(enumLibs);

DWORD fetched;

IVsLibrary2 lib;

while(enumLibs.Next(1, &lib, &fetched) == S\_OK && fetched == 1)

{

scope(exit) release(lib);

if(auto slib = qi\_cast!IVsSimpleLibrary2(lib))

{

scope(exit) release(slib);

IVsSimpleObjectList2 reslist;

if(slib.GetList2(LLT\_MEMBERS, LLF\_USESEARCHFILTER, &searchOpts, &reslist) == S\_OK)

{

scope(exit) release(reslist);

ULONG items;

if(reslist.GetItemCount(&items) == S\_OK && items > 0)

{

BOOL ok;

for(ULONG it = 0; it < items; it++)

if(reslist.CanGoToSource(it, GS\_DEFINITION, &ok) == S\_OK && ok)

if(reslist.GoToSource(it, GS\_DEFINITION) == S\_OK)

return S\_OK;

}

}

}

}

}

}

}

return S\_FALSE;

}

else

HRESULT GotoDefinitionCPP()

{

if(auto navmgr = queryService!(SVsSymbolicNavigationManager, IVsSymbolicNavigationManager))

{

scope(exit) release(navmgr);

string word = toUTF8(GetWordAtCaret());

IVsUIShellOpenDocument pIVsUIShellOpenDocument = queryService!(IVsUIShellOpenDocument);

if(!pIVsUIShellOpenDocument)

return returnError(E\_FAIL);

scope(exit) release(pIVsUIShellOpenDocument);

string fname = mCodeWinMgr.mSource.GetFileName();

wchar\* wfname = \_toUTF16z(fname);

string addopt;

IVsUIHierarchy pUIH;

uint itemid;

IServiceProvider pSP;

VSDOCINPROJECT docInProj;

if(pIVsUIShellOpenDocument.IsDocumentInAProject(wfname, &pUIH, &itemid, &pSP, &docInProj) != S\_OK)

return S\_OK;

scope(exit) release(pSP);

scope(exit) release(pUIH);

if(!pUIH)

return returnError(E\_FAIL);

Project proj = qi\_cast!Project(pUIH);

scope(exit) release(proj);

BOOL handled;

HRESULT hr = navmgr.OnBeforeNavigateToSymbol(proj, itemid, \_toUTF16z(word), &handled);

if(hr == S\_OK && handled)

return hr;

}

return S\_FALSE;

}

HRESULT GotoDefinitionJSON(string file, string word)

{

Definition[] defs = Package.GetLibInfos().findDefinition(word);

if(defs.length == 0)

{

showStatusBarText("No definition found for '" ~ word ~ "'");

return S\_FALSE;

}

if(defs.length > 1)

{

showStatusBarText("Multiple definitions found for '" ~ word ~ "'");

showSearchWindow(false, word);

return S\_FALSE;

}

string deffile = defs[0].filename;

int defline = defs[0].line;

HRESULT hr = OpenFileInSolution(deffile, defline, 0, file, true);

if(hr != S\_OK)

showStatusBarText(format("Cannot open %s(%d) for definition of '%s'", deffile, defline, word));

return hr;

}

extern(D)

void GotoDefinitionCallBack(uint request, string fname, sdk.vsi.sdk\_shared.TextSpan span)

{

bool extrn = fname.startsWith("EXTERN:");

if (extrn)

fname = fname[7..$];

if (extrn && !mLastGotoDecl)

{

string word = split(mLastGotoDef, ".")[$-1];

if(GotoDefinitionCPP(word) == S\_OK)

return;

}

if (fname.length)

{

HRESULT hr = OpenFileInSolution(fname, span.iStartLine, span.iStartIndex, null, false);

if(hr != S\_OK)

showStatusBarText(format("Cannot open %s(%d) for goto definition", fname, span.iStartLine));

}

else

{

string word = split(mLastGotoDef, ".")[$-1];

GotoDefinitionJSON(mCodeWinMgr.mSource.GetFileName(), word);

//showStatusBarText("No definition found for '" ~ mLastGotoDef ~ "'");

}

}

//////////////////////////////////////////////////////////////

int HandleFindReferences()

{

int line, idx;

if(mView.GetCaretPos(&line, &idx) != S\_OK)

return S\_FALSE;

Package.GetLanguageService().GetReferences(mCodeWinMgr.mSource, "", line, idx, &FindReferencesCallBack);

return S\_FALSE;

}

extern(D)

void FindReferencesCallBack(uint request, string filename, string tok, int line, int idx, string[] exps)

{

if(IVsFindSymbol fs = queryService!(IVsObjectSearch, IVsFindSymbol))

{

scope(exit) release(fs);

VSOBSEARCHCRITERIA2 criteria;

criteria.dwCustom = 0; // FindReferencesResults;

criteria.eSrchType = SO\_ENTIREWORD,

criteria.grfOptions = VSOBSO\_LISTREFERENCES,

criteria.pIVsNavInfo = null,

criteria.szName = "Find All References";

if (auto lib = Package.s\_instance.GetLibrary())

lib.mLastFindReferencesResult = exps;

fs.DoSearch(&GUID\_VsSymbolScope\_All, &criteria);

}

}

//////////////////////////////////////////////////////////////

int HandleHelp()

{

string word = toUTF8(GetWordAtCaret());

if(word.length <= 0)

return S\_FALSE;

if(!openHelp(word))

showStatusBarText(text("No documentation found for '", word, "'."));

return S\_OK;

}

//////////////////////////////////////////////////////////////

int HandleMethodTip()

{

int rc = \_HandleMethodTip();

if(rc != S\_OK)

mCodeWinMgr.mSource.DismissMethodTip();

return rc;

}

int \_HandleMethodTip(bool tryUpper = true)

{

TextSpan span;

if(mView.GetCaretPos(&span.iStartLine, &span.iStartIndex) != S\_OK)

return S\_FALSE;

int line = span.iStartLine;

int idx = span.iStartIndex;

int iState;

uint pos;

int tok = mCodeWinMgr.mSource.FindLineToken(line, idx, iState, pos);

stepUp:

int otherLine, otherIndex, cntComma;

Source src = mCodeWinMgr.mSource;

if(!src.FindOpeningBracketBackward(line, tok, otherLine, otherIndex, &cntComma))

return S\_FALSE;

wstring bracket = src.GetText(otherLine, otherIndex, otherLine, otherIndex + 1);

if(bracket != "("w)

return S\_FALSE;

tok = mCodeWinMgr.mSource.FindLineToken(otherLine, otherIndex, iState, pos);

string word = toUTF8(src.FindMethodIdentifierBackward(otherLine, tok, &line, &idx));

if(word.length <= 0)

{

line = otherLine;

idx = otherIndex;

if(!tryUpper)

return S\_FALSE;

goto stepUp;

}

span.iStartIndex = idx;

span.iStartLine = line;

span.iEndIndex = idx + 1;

span.iEndLine = line;

mPendingMethodTipWord = word;

mPendingMethodTipComma = cntComma;

mPendingRequest = Package.GetLanguageService().GetTip(mCodeWinMgr.mSource, &span, &OnGetMethodTipText);

return S\_OK;

}

extern(D) void OnGetMethodTipText(uint request, string filename, string text, TextSpan span)

{

Definition[] defs;

string[] funcs = split(text, "\a");

if(funcs.empty)

{

defs = Package.GetLibInfos().findDefinition(mPendingMethodTipWord);

}

else

{

foreach(fn; funcs)

{

Definition def;

def.name = mPendingMethodTipWord;

int pos = fn.indexOf("\n");

if(pos >= 0)

{

def.help = fn[pos + 1 .. $];

if(def.help.startsWith("(Deduced Type"))

if(!findSkip(def.help, "\n"))

def.help = "";

fn = fn[0 .. pos];

}

if(fn.endsWith("\r"))

fn = fn[0..$-1];

if(fn.endsWith(":"))

fn = fn[0..$-1];

def.setType(fn);

defs ~= def;

}

}

RefreshMethodTip(defs, span);

}

int RefreshMethodTip(Definition[] defs, TextSpan span)

{

if(defs.length == 0)

return S\_FALSE;

MethodData md = mCodeWinMgr.mSource.GetMethodData();

span.iEndLine = span.iStartLine;

span.iEndIndex = span.iStartIndex + 1;

md.Refresh(mView, defs, mPendingMethodTipComma, span);

return S\_OK;

}

// IVsTextViewFilter //////////////////////////////////////

override int GetWordExtent(in int iLine, in CharIndex iIndex, in uint dwFlags, /\* [out] \*/ TextSpan \*pSpan)

{

mixin(LogCallMix);

int startIdx, endIdx;

if(!mCodeWinMgr.mSource.GetWordExtent(iLine, iIndex, dwFlags, startIdx, endIdx))

return S\_FALSE;

pSpan.iStartLine = iLine;

pSpan.iStartIndex = startIdx;

pSpan.iEndLine = iLine;

pSpan.iEndIndex = endIdx;

return S\_OK;

}

override int GetDataTipText( /\* [out][in] \*/ TextSpan \*pSpan, /\* [out] \*/ BSTR \*pbstrText)

{

mixin(LogCallMix);

HRESULT resFwd = TIP\_S\_ONLYIFNOMARKER; // enable and prefer TextMarker tooltips

TextSpan span = \*pSpan;

if(!mCodeWinMgr.mSource.GetTipSpan(pSpan))

return resFwd;

// when implementing IVsTextViewFilter, VS2010 will no longer ask the debugger

// for tooltips, so we have to do it ourselves

if(IVsDebugger srpVsDebugger = queryService!(IVsDebugger))

{

scope(exit) release(srpVsDebugger);

HRESULT hr = srpVsDebugger.GetDataTipValue(mCodeWinMgr.mSource.mBuffer, pSpan, null, pbstrText);

if(hr == 0x45001) // always returned when debugger active, so no other tooltips then

{

if(IVsCustomDataTip tip = qi\_cast!IVsCustomDataTip(srpVsDebugger))

{

scope(exit) release(tip);

if(SUCCEEDED (tip.DisplayDataTip()))

return S\_OK;

}

else

return hr;

} // return hr; // this triggers HandoffNoDefaultTipToDebugger

}

version(none) // quick info tooltips not good enough yet

{

string word = toUTF8(mCodeWinMgr.mSource.GetText(pSpan.iStartLine, pSpan.iStartIndex, pSpan.iEndLine, pSpan.iEndIndex));

if(word.length <= 0)

return resFwd;

Definition[] defs = Package.GetLibInfos().findDefinition(word);

if(defs.length == 0)

return resFwd;

string msg = word;

foreach(def; defs)

{

string m = "\n" ~ def.kind ~ "\t" ~ def.filename;

if(def.line > 0)

m ~= ":" ~ to!(string)(def.line);

msg ~= m;

}

\*pbstrText = allocBSTR(msg);

}

if(Package.GetGlobalOptions().showTypeInTooltip)

{

if(mPendingSpan == span && mTipRequest == mPendingRequest)

{

\*pbstrText = allocBSTR(mTipText);

\*pSpan = mTipSpan;

}

else

{

if(mPendingSpan != span)

{

mPendingSpan = span;

mPendingRequest = Package.GetLanguageService().GetTip(mCodeWinMgr.mSource, &span, &OnGetTipText);

}

return E\_PENDING;

}

}

return resFwd;

}

override int GetPairExtents(in int iLine, in CharIndex iIndex, /\* [out] \*/ TextSpan \*pSpan)

{

mixin(LogCallMix);

return E\_NOTIMPL;

}

// IVsTextViewEvents //////////////////////////////////////

override int OnSetFocus(IVsTextView pView)

{

mixin(LogCallMix);

mCodeWinMgr.mLangSvc.mLastActiveView = this;

return S\_OK;

}

override int OnKillFocus(IVsTextView pView)

{

mixin(LogCallMix);

if(mCodeWinMgr.mLangSvc.mLastActiveView is this)

mCodeWinMgr.mLangSvc.mLastActiveView = null;

return S\_OK;

}

override int OnSetBuffer(IVsTextView pView, IVsTextLines pBuffer)

{

mixin(LogCallMix);

return S\_OK;

}

override int OnChangeScrollInfo(IVsTextView pView, in int iBar,

in int iMinUnit, in int iMaxUnits,

in int iVisibleUnits, in int iFirstVisibleUnit)

{

// mixin(LogCallMix);

return S\_OK;

}

override int OnChangeCaretLine(IVsTextView pView, in int iNewLine, in int iOldLine)

{

// mixin(LogCallMix);

return S\_OK;

}

// IVsExpansionEvents //////////////////////////////////////

override int OnAfterSnippetsUpdate()

{

mixin(LogCallMix);

return S\_OK;

}

override int OnAfterSnippetsKeyBindingChange(in uint dwCmdGuid, in uint dwCmdId, in BOOL fBound)

{

mixin(LogCallMix);

return S\_OK;

}

//////////////////////////////

TextSpan mPendingSpan;

uint mPendingRequest;

int mPendingMethodTipComma;

string mPendingMethodTipWord;

TextSpan mTipSpan;

string mTipText;

uint mTipRequest;

string mLastGotoDef;

bool mLastGotoDecl;

extern(D) void OnGetTipText(uint request, string filename, string text, TextSpan span)

{

text = replace(text, "\a", "\n\n");

mTipText = phobosDdocExpand(text);

mTipSpan = span;

mTipRequest = request;

version(tip)

{

mTextTipData.Init(mView, "Huu: " ~ text);

mTextTipData.UpdateView();

}

}

bool OnIdle()

{

int line;

ViewCol idx;

if(int rc = mView.GetCaretPos(&line, &idx))

return false;

if(mLastHighlightBracesLine == line && mLastHighlightBracesCol == idx)

return false;

mLastHighlightBracesLine = line;

mLastHighlightBracesCol = idx;

HighlightMatchingPairs();

version(tip)

{

string msg = mCodeWinMgr.mSource.getParseError(line, idx);

if(msg.length)

{

mTextTipData.Init(mView, msg);

mTextTipData.UpdateView();

}

else

mTextTipData.Dismiss();

}

return true;

}

}

class TextTipData : DisposingComObject, IVsTextTipData

{

IVsTextTipWindow mTipWindow;

IVsTextView mTextView;

string mTipText;

bool mDisplayed;

this()

{

mTipText = "Tipp";

auto uuid = uuid\_coclass\_VsTextTipWindow;

mTipWindow = VsLocalCreateInstance!IVsTextTipWindow (&uuid, sdk.win32.wtypes.CLSCTX\_INPROC\_SERVER);

if (mTipWindow)

mTipWindow.SetTextTipData(this);

}

override HRESULT QueryInterface(in IID\* riid, void\*\* pvObject)

{

if(queryInterface!(IVsTextTipData) (this, riid, pvObject))

return S\_OK;

return super.QueryInterface(riid, pvObject);

}

void Init(IVsTextView textView, string tip)

{

Close();

mTextView = textView;

mTipText = tip;

mDisplayed = false;

}

void Close()

{

Dismiss();

}

void Dismiss()

{

if (mDisplayed && mTextView)

mTextView.UpdateTipWindow(mTipWindow, UTW\_DISMISS);

OnDismiss();

}

override void Dispose()

{

Close();

if (mTipWindow)

mTipWindow.SetTextTipData(null);

mTipWindow = release(mTipWindow);

}

HRESULT GetTipText(/+[out, custom(uuid\_IVsTextTipData, "optional")]+/ BSTR \*pbstrText,

/+[out]+/ BOOL \*pfGetFontInfo)

{

if(pbstrText)

\*pbstrText = allocBSTR(mTipText);

if(pfGetFontInfo)

\*pfGetFontInfo = FALSE;

return S\_OK;

}

// NOTE: \*pdwFontAttr will already have been memset-ed to zeroes, so you can set only the indices that are not normal

HRESULT GetTipFontInfo(in int cChars, /+[out, size\_is(cChars)]+/ ULONG \*pdwFontAttr)

{

// needs \*pfGetFontInfo = TRUE; above

// 1 for bold

return E\_NOTIMPL;

}

HRESULT GetContextStream(/+[out]+/ int \*piPos, /+[out]+/ int \*piLength)

{

int line, idx, vspace, endpos;

if(HRESULT rc = mTextView.GetCaretPos(&line, &idx))

return rc;

if(HRESULT rc = mTextView.GetNearestPosition(line, idx, piPos, &vspace))

return rc;

\*piLength = 1;

return S\_OK;

}

HRESULT OnDismiss()

{

mTextView = null;

mDisplayed = false;

return S\_OK;

}

HRESULT UpdateView()

{

if (mTextView && mTipWindow)

{

mTextView.UpdateTipWindow(mTipWindow, UTW\_CONTENTCHANGED);

mDisplayed = true;

}

return S\_OK;

}

}

// This file is part of Visual D

//

// Visual D integrates the D programming language into Visual Studio

// Copyright (c) 2010 by Rainer Schuetze, All Rights Reserved

//

// Distributed under the Boost Software License, Version 1.0.

// See accompanying file LICENSE\_1\_0.txt or copy at <http://www.boost.org/LICENSE_1_0.txt>

module visuald.vscommands;

import visuald.windows;

import std.string;

import visuald.logutil;

import sdk.vsi.stdidcmd;

import sdk.vsi.vsshell;

import sdk.vsi.vsdebugguids;

import sdk.vsi.vsdbgcmd;

import sdk.vsi.sdk\_version;

debug

{

static if (sdk.vsi.sdk\_version.rmj > 9)

{

alias cmdidShellNavigate1First cmdidShellNavigate1;

alias cmdidShellNavigate2First cmdidShellNavigate2;

alias cmdidShellNavigate3First cmdidShellNavigate3;

alias cmdidShellNavigate4First cmdidShellNavigate4;

alias cmdidShellNavigate5First cmdidShellNavigate5;

alias cmdidShellNavigate6First cmdidShellNavigate6;

alias cmdidShellNavigate7First cmdidShellNavigate7;

alias cmdidShellNavigate8First cmdidShellNavigate8;

alias cmdidShellNavigate9First cmdidShellNavigate9;

alias cmdidShellNavigate10First cmdidShellNavigate10;

alias cmdidShellNavigate11First cmdidShellNavigate11;

alias cmdidShellNavigate12First cmdidShellNavigate12;

alias cmdidShellNavigate13First cmdidShellNavigate13;

alias cmdidShellNavigate14First cmdidShellNavigate14;

alias cmdidShellNavigate15First cmdidShellNavigate15;

alias cmdidShellNavigate16First cmdidShellNavigate16;

alias cmdidShellNavigate17First cmdidShellNavigate17;

alias cmdidShellNavigate18First cmdidShellNavigate18;

alias cmdidShellNavigate19First cmdidShellNavigate19;

alias cmdidShellNavigate20First cmdidShellNavigate20;

alias cmdidShellNavigate21First cmdidShellNavigate21;

alias cmdidShellNavigate22First cmdidShellNavigate22;

alias cmdidShellNavigate23First cmdidShellNavigate23;

alias cmdidShellNavigate24First cmdidShellNavigate24;

alias cmdidShellNavigate25First cmdidShellNavigate25;

alias cmdidShellNavigate26First cmdidShellNavigate26;

alias cmdidShellNavigate27First cmdidShellNavigate27;

alias cmdidShellNavigate28First cmdidShellNavigate28;

alias cmdidShellNavigate29First cmdidShellNavigate29;

alias cmdidShellNavigate30First cmdidShellNavigate30;

alias cmdidShellNavigate31First cmdidShellNavigate31;

alias cmdidShellNavigate32First cmdidShellNavigate32;

alias cmdidShellNavigateLast cmdidShellNavigate33;

enum ECMD\_PROMOTELOCAL = 1554; // no longer in SDK 10.0

enum cmdidProgramToDebugShow = 0x00000102;

}

struct enumName

{

uint id;

string name;

}

const enumName[] VSStd2KCmdID\_names =

[

{ ECMD\_TYPECHAR, "TYPECHAR" },

{ ECMD\_BACKSPACE, "BACKSPACE" },

{ ECMD\_RETURN, "RETURN" },

{ ECMD\_TAB, "ECMD\_TAB" },

{ ECMD\_TAB, "TAB" },

{ ECMD\_BACKTAB, "BACKTAB" },

{ ECMD\_DELETE, "DELETE" },

{ ECMD\_LEFT, "LEFT" },

{ ECMD\_LEFT\_EXT, "LEFT\_EXT" },

{ ECMD\_RIGHT, "RIGHT" },

{ ECMD\_RIGHT\_EXT, "RIGHT\_EXT" },

{ ECMD\_UP, "UP" },

{ ECMD\_UP\_EXT, "UP\_EXT" },

{ ECMD\_DOWN, "DOWN" },

{ ECMD\_DOWN\_EXT, "DOWN\_EXT" },

{ ECMD\_HOME, "HOME" },

{ ECMD\_HOME\_EXT, "HOME\_EXT" },

{ ECMD\_END, "END" },

{ ECMD\_END\_EXT, "END\_EXT" },

{ ECMD\_BOL, "BOL" },

{ ECMD\_BOL\_EXT, "BOL\_EXT" },

{ ECMD\_FIRSTCHAR, "FIRSTCHAR" },

{ ECMD\_FIRSTCHAR\_EXT, "FIRSTCHAR\_EXT" },

{ ECMD\_EOL, "EOL" },

{ ECMD\_EOL\_EXT, "EOL\_EXT" },

{ ECMD\_LASTCHAR, "LASTCHAR" },

{ ECMD\_LASTCHAR\_EXT, "LASTCHAR\_EXT" },

{ ECMD\_PAGEUP, "PAGEUP" },

{ ECMD\_PAGEUP\_EXT, "PAGEUP\_EXT" },

{ ECMD\_PAGEDN, "PAGEDN" },

{ ECMD\_PAGEDN\_EXT, "PAGEDN\_EXT" },

{ ECMD\_TOPLINE, "TOPLINE" },

{ ECMD\_TOPLINE\_EXT, "TOPLINE\_EXT" },

{ ECMD\_BOTTOMLINE, "BOTTOMLINE" },

{ ECMD\_BOTTOMLINE\_EXT, "BOTTOMLINE\_EXT" },

{ ECMD\_SCROLLUP, "SCROLLUP" },

{ ECMD\_SCROLLDN, "SCROLLDN" },

{ ECMD\_SCROLLPAGEUP, "SCROLLPAGEUP" },

{ ECMD\_SCROLLPAGEDN, "SCROLLPAGEDN" },

{ ECMD\_SCROLLLEFT, "SCROLLLEFT" },

{ ECMD\_SCROLLRIGHT, "SCROLLRIGHT" },

{ ECMD\_SCROLLBOTTOM, "SCROLLBOTTOM" },

{ ECMD\_SCROLLCENTER, "SCROLLCENTER" },

{ ECMD\_SCROLLTOP, "SCROLLTOP" },

{ ECMD\_SELECTALL, "SELECTALL" },

{ ECMD\_SELTABIFY, "SELTABIFY" },

{ ECMD\_SELUNTABIFY, "SELUNTABIFY" },

{ ECMD\_SELLOWCASE, "SELLOWCASE" },

{ ECMD\_SELUPCASE, "SELUPCASE" },

{ ECMD\_SELTOGGLECASE, "SELTOGGLECASE" },

{ ECMD\_SELTITLECASE, "SELTITLECASE" },

{ ECMD\_SELSWAPANCHOR, "SELSWAPANCHOR" },

{ ECMD\_GOTOLINE, "GOTOLINE" },

{ ECMD\_GOTOBRACE, "GOTOBRACE" },

{ ECMD\_GOTOBRACE\_EXT, "GOTOBRACE\_EXT" },

{ ECMD\_GOBACK, "GOBACK" },

{ ECMD\_SELECTMODE, "SELECTMODE" },

{ ECMD\_TOGGLE\_OVERTYPE\_MODE, "TOGGLE\_OVERTYPE\_MODE" },

{ ECMD\_CUT, "CUT" },

{ ECMD\_COPY, "COPY" },

{ ECMD\_PASTE, "PASTE" },

{ ECMD\_CUTLINE, "CUTLINE" },

{ ECMD\_DELETELINE, "DELETELINE" },

{ ECMD\_DELETEBLANKLINES, "DELETEBLANKLINES" },

{ ECMD\_DELETEWHITESPACE, "DELETEWHITESPACE" },

{ ECMD\_DELETETOEOL, "DELETETOEOL" },

{ ECMD\_DELETETOBOL, "DELETETOBOL" },

{ ECMD\_OPENLINEABOVE, "OPENLINEABOVE" },

{ ECMD\_OPENLINEBELOW, "OPENLINEBELOW" },

{ ECMD\_INDENT, "INDENT" },

{ ECMD\_UNINDENT, "UNINDENT" },

{ ECMD\_UNDO, "UNDO" },

{ ECMD\_UNDONOMOVE, "UNDONOMOVE" },

{ ECMD\_REDO, "REDO" },

{ ECMD\_REDONOMOVE, "REDONOMOVE" },

{ ECMD\_DELETEALLTEMPBOOKMARKS, "DELETEALLTEMPBOOKMARKS" },

{ ECMD\_TOGGLETEMPBOOKMARK, "TOGGLETEMPBOOKMARK" },

{ ECMD\_GOTONEXTBOOKMARK, "GOTONEXTBOOKMARK" },

{ ECMD\_GOTOPREVBOOKMARK, "GOTOPREVBOOKMARK" },

{ ECMD\_FIND, "FIND" },

{ ECMD\_REPLACE, "REPLACE" },

{ ECMD\_REPLACE\_ALL, "REPLACE\_ALL" },

{ ECMD\_FINDNEXT, "FINDNEXT" },

{ ECMD\_FINDNEXTWORD, "FINDNEXTWORD" },

{ ECMD\_FINDPREV, "FINDPREV" },

{ ECMD\_FINDPREVWORD, "FINDPREVWORD" },

{ ECMD\_FINDAGAIN, "FINDAGAIN" },

{ ECMD\_TRANSPOSECHAR, "TRANSPOSECHAR" },

{ ECMD\_TRANSPOSEWORD, "TRANSPOSEWORD" },

{ ECMD\_TRANSPOSELINE, "TRANSPOSELINE" },

{ ECMD\_SELECTCURRENTWORD, "SELECTCURRENTWORD" },

{ ECMD\_DELETEWORDRIGHT, "DELETEWORDRIGHT" },

{ ECMD\_DELETEWORDLEFT, "DELETEWORDLEFT" },

{ ECMD\_WORDPREV, "WORDPREV" },

{ ECMD\_WORDPREV\_EXT, "WORDPREV\_EXT" },

{ ECMD\_WORDNEXT, "WORDNEXT" },

{ ECMD\_WORDNEXT\_EXT, "WORDNEXT\_EXT" },

{ ECMD\_COMMENTBLOCK, "COMMENTBLOCK" },

{ ECMD\_UNCOMMENTBLOCK, "UNCOMMENTBLOCK" },

{ ECMD\_SETREPEATCOUNT, "SETREPEATCOUNT" },

{ ECMD\_WIDGETMARGIN\_LBTNDOWN, "WIDGETMARGIN\_LBTNDOWN" },

{ ECMD\_SHOWCONTEXTMENU, "SHOWCONTEXTMENU" },

{ ECMD\_CANCEL, "CANCEL" },

{ ECMD\_PARAMINFO, "PARAMINFO" },

{ ECMD\_TOGGLEVISSPACE, "TOGGLEVISSPACE" },

{ ECMD\_TOGGLECARETPASTEPOS, "TOGGLECARETPASTEPOS" },

{ ECMD\_COMPLETEWORD, "COMPLETEWORD" },

{ ECMD\_SHOWMEMBERLIST, "SHOWMEMBERLIST" },

{ ECMD\_FIRSTNONWHITEPREV, "FIRSTNONWHITEPREV" },

{ ECMD\_FIRSTNONWHITENEXT, "FIRSTNONWHITENEXT" },

{ ECMD\_HELPKEYWORD, "HELPKEYWORD" },

{ ECMD\_FORMATSELECTION, "FORMATSELECTION" },

{ ECMD\_OPENURL, "OPENURL" },

{ ECMD\_INSERTFILE, "INSERTFILE" },

{ ECMD\_TOGGLESHORTCUT, "TOGGLESHORTCUT" },

{ ECMD\_QUICKINFO, "QUICKINFO" },

{ ECMD\_LEFT\_EXT\_COL, "LEFT\_EXT\_COL" },

{ ECMD\_RIGHT\_EXT\_COL, "RIGHT\_EXT\_COL" },

{ ECMD\_UP\_EXT\_COL, "UP\_EXT\_COL" },

{ ECMD\_DOWN\_EXT\_COL, "DOWN\_EXT\_COL" },

{ ECMD\_TOGGLEWORDWRAP, "TOGGLEWORDWRAP" },

{ ECMD\_ISEARCH, "ISEARCH" },

{ ECMD\_ISEARCHBACK, "ISEARCHBACK" },

{ ECMD\_BOL\_EXT\_COL, "BOL\_EXT\_COL" },

{ ECMD\_EOL\_EXT\_COL, "EOL\_EXT\_COL" },

{ ECMD\_WORDPREV\_EXT\_COL, "WORDPREV\_EXT\_COL" },

{ ECMD\_WORDNEXT\_EXT\_COL, "WORDNEXT\_EXT\_COL" },

{ ECMD\_OUTLN\_HIDE\_SELECTION, "OUTLN\_HIDE\_SELECTION" },

{ ECMD\_OUTLN\_TOGGLE\_CURRENT, "OUTLN\_TOGGLE\_CURRENT" },

{ ECMD\_OUTLN\_TOGGLE\_ALL, "OUTLN\_TOGGLE\_ALL" },

{ ECMD\_OUTLN\_STOP\_HIDING\_ALL, "OUTLN\_STOP\_HIDING\_ALL" },

{ ECMD\_OUTLN\_STOP\_HIDING\_CURRENT, "OUTLN\_STOP\_HIDING\_CURRENT" },

{ ECMD\_OUTLN\_COLLAPSE\_TO\_DEF, "OUTLN\_COLLAPSE\_TO\_DEF" },

{ ECMD\_DOUBLECLICK, "DOUBLECLICK" },

{ ECMD\_EXTERNALLY\_HANDLED\_WIDGET\_CLICK, "EXTERNALLY\_HANDLED\_WIDGET\_CLICK" },

{ ECMD\_COMMENT\_BLOCK, "COMMENT\_BLOCK" },

{ ECMD\_UNCOMMENT\_BLOCK, "UNCOMMENT\_BLOCK" },

{ ECMD\_OPENFILE, "OPENFILE" },

{ ECMD\_NAVIGATETOURL, "NAVIGATETOURL" },

{ ECMD\_HANDLEIMEMESSAGE, "HANDLEIMEMESSAGE" },

{ ECMD\_SELTOGOBACK, "SELTOGOBACK" },

{ ECMD\_COMPLETION\_HIDE\_ADVANCED, "COMPLETION\_HIDE\_ADVANCED" },

{ ECMD\_FORMATDOCUMENT, "FORMATDOCUMENT" },

{ ECMD\_OUTLN\_START\_AUTOHIDING, "OUTLN\_START\_AUTOHIDING" },

{ ECMD\_FINAL, "FINAL" },

{ ECMD\_DECREASEFILTER, "ECMD\_DECREASEFILTER" },

{ ECMD\_COPYTIP, "ECMD\_COPYTIP" },

{ ECMD\_PASTETIP, "ECMD\_PASTETIP" },

{ ECMD\_LEFTCLICK, "ECMD\_LEFTCLICK" },

{ ECMD\_GOTONEXTBOOKMARKINDOC, "ECMD\_GOTONEXTBOOKMARKINDOC" },

{ ECMD\_GOTOPREVBOOKMARKINDOC, "ECMD\_GOTOPREVBOOKMARKINDOC" },

{ ECMD\_INVOKESNIPPETFROMSHORTCUT, "ECMD\_INVOKESNIPPETFROMSHORTCUT" },

{ ECMD\_AUTOCOMPLETE, "AUTOCOMPLETE" },

{ ECMD\_INVOKESNIPPETPICKER2, "ECMD\_INVOKESNIPPETPICKER2" },

{ ECMD\_DELETEALLBOOKMARKSINDOC, "ECMD\_DELETEALLBOOKMARKSINDOC" },

{ ECMD\_CONVERTTABSTOSPACES, "ECMD\_CONVERTTABSTOSPACES" },

{ ECMD\_CONVERTSPACESTOTABS, "ECMD\_CONVERTSPACESTOTABS" },

{ ECMD\_FINAL, "ECMD\_FINAL" },

{ ECMD\_STOP, "STOP" },

{ ECMD\_REVERSECANCEL, "REVERSECANCEL" },

{ ECMD\_SLNREFRESH, "SLNREFRESH" },

{ ECMD\_SAVECOPYOFITEMAS, "SAVECOPYOFITEMAS" },

{ ECMD\_NEWELEMENT, "NEWELEMENT" },

{ ECMD\_NEWATTRIBUTE, "NEWATTRIBUTE" },

{ ECMD\_NEWCOMPLEXTYPE, "NEWCOMPLEXTYPE" },

{ ECMD\_NEWSIMPLETYPE, "NEWSIMPLETYPE" },

{ ECMD\_NEWGROUP, "NEWGROUP" },

{ ECMD\_NEWATTRIBUTEGROUP, "NEWATTRIBUTEGROUP" },

{ ECMD\_NEWKEY, "NEWKEY" },

{ ECMD\_NEWRELATION, "NEWRELATION" },

{ ECMD\_EDITKEY, "EDITKEY" },

{ ECMD\_EDITRELATION, "EDITRELATION" },

{ ECMD\_MAKETYPEGLOBAL, "MAKETYPEGLOBAL" },

{ ECMD\_PREVIEWDATASET, "PREVIEWDATASET" },

{ ECMD\_GENERATEDATASET, "GENERATEDATASET" },

{ ECMD\_CREATESCHEMA, "CREATESCHEMA" },

{ ECMD\_LAYOUTINDENT, "LAYOUTINDENT" },

{ ECMD\_LAYOUTUNINDENT, "LAYOUTUNINDENT" },

{ ECMD\_REMOVEHANDLER, "REMOVEHANDLER" },

{ ECMD\_EDITHANDLER, "EDITHANDLER" },

{ ECMD\_ADDHANDLER, "ADDHANDLER" },

//        { ECMD\_STYLE, "STYLE" },

//        { ECMD\_STYLEGETLIST, "STYLEGETLIST" },

{ ECMD\_FONTSTYLE, "FONTSTYLE" },

{ ECMD\_FONTSTYLEGETLIST, "FONTSTYLEGETLIST" },

{ ECMD\_PASTEASHTML, "PASTEASHTML" },

{ ECMD\_VIEWBORDERS, "VIEWBORDERS" },

{ ECMD\_VIEWDETAILS, "VIEWDETAILS" },

//        { ECMD\_EXPANDCONTROLS, "EXPANDCONTROLS" },

//        { ECMD\_COLLAPSECONTROLS, "COLLAPSECONTROLS" },

//        { ECMD\_SHOWSCRIPTONLY, "SHOWSCRIPTONLY" },

{ ECMD\_INSERTTABLE, "INSERTTABLE" },

{ ECMD\_INSERTCOLLEFT, "INSERTCOLLEFT" },

{ ECMD\_INSERTCOLRIGHT, "INSERTCOLRIGHT" },

{ ECMD\_INSERTROWABOVE, "INSERTROWABOVE" },

{ ECMD\_INSERTROWBELOW, "INSERTROWBELOW" },

{ ECMD\_DELETETABLE, "DELETETABLE" },

{ ECMD\_DELETECOLS, "DELETECOLS" },

{ ECMD\_DELETEROWS, "DELETEROWS" },

{ ECMD\_SELECTTABLE, "SELECTTABLE" },

{ ECMD\_SELECTTABLECOL, "SELECTTABLECOL" },

{ ECMD\_SELECTTABLEROW, "SELECTTABLEROW" },

{ ECMD\_SELECTTABLECELL, "SELECTTABLECELL" },

{ ECMD\_MERGECELLS, "MERGECELLS" },

{ ECMD\_SPLITCELL, "SPLITCELL" },

//        { ECMD\_INSERTCELL, "INSERTCELL" },

{ ECMD\_DELETECELLS, "DELETECELLS" },

//        { ECMD\_SEAMLESSFRAME, "SEAMLESSFRAME" },

//        { ECMD\_VIEWFRAME, "VIEWFRAME" },

//        { ECMD\_DELETEFRAME, "DELETEFRAME" },

//        { ECMD\_SETFRAMESOURCE, "SETFRAMESOURCE" },

//        { ECMD\_NEWLEFTFRAME, "NEWLEFTFRAME" },

//        { ECMD\_NEWRIGHTFRAME, "NEWRIGHTFRAME" },

//        { ECMD\_NEWTOPFRAME, "NEWTOPFRAME" },

//        { ECMD\_NEWBOTTOMFRAME, "NEWBOTTOMFRAME" },

{ ECMD\_SHOWGRID, "SHOWGRID" },

{ ECMD\_SNAPTOGRID, "SNAPTOGRID" },

{ ECMD\_BOOKMARK, "BOOKMARK" },

{ ECMD\_HYPERLINK, "HYPERLINK" },

//        { ECMD\_IMAGE, "IMAGE" },

//        { ECMD\_INSERTFORM, "INSERTFORM" },

//        { ECMD\_INSERTSPAN, "INSERTSPAN" },

//        { ECMD\_DIV, "DIV" },

//        { ECMD\_HTMLCLIENTSCRIPTBLOCK, "HTMLCLIENTSCRIPTBLOCK" },

//        { ECMD\_HTMLSERVERSCRIPTBLOCK, "HTMLSERVERSCRIPTBLOCK" },

{ ECMD\_BULLETEDLIST, "BULLETEDLIST" },

{ ECMD\_NUMBEREDLIST, "NUMBEREDLIST" },

{ ECMD\_EDITSCRIPT, "EDITSCRIPT" },

{ ECMD\_EDITCODEBEHIND, "EDITCODEBEHIND" },

{ ECMD\_DOCOUTLINEHTML, "DOCOUTLINEHTML" },

//        { ECMD\_DOCOUTLINESCRIPT, "DOCOUTLINESCRIPT" },

{ ECMD\_RUNATSERVER, "RUNATSERVER" },

{ ECMD\_WEBFORMSVERBS, "WEBFORMSVERBS" },

{ ECMD\_WEBFORMSTEMPLATES, "WEBFORMSTEMPLATES" },

{ ECMD\_ENDTEMPLATE, "ENDTEMPLATE" },

{ ECMD\_EDITDEFAULTEVENT, "EDITDEFAULTEVENT" },

{ ECMD\_SUPERSCRIPT, "SUPERSCRIPT" },

{ ECMD\_SUBSCRIPT, "SUBSCRIPT" },

{ ECMD\_EDITSTYLE, "EDITSTYLE" },

{ ECMD\_ADDIMAGEHEIGHTWIDTH, "ADDIMAGEHEIGHTWIDTH" },

{ ECMD\_REMOVEIMAGEHEIGHTWIDTH, "REMOVEIMAGEHEIGHTWIDTH" },

{ ECMD\_LOCKELEMENT, "LOCKELEMENT" },

//        { ECMD\_VIEWSTYLEORGANIZER, "VIEWSTYLEORGANIZER" },

{ ECMD\_AUTOCLOSEOVERRIDE, "ECMD\_AUTOCLOSEOVERRIDE" },

{ ECMD\_NEWANY, "NEWANY" },

{ ECMD\_NEWANYATTRIBUTE, "NEWANYATTRIBUTE" },

{ ECMD\_DELETEKEY, "DELETEKEY" },

{ ECMD\_AUTOARRANGE, "AUTOARRANGE" },

{ ECMD\_VALIDATESCHEMA, "VALIDATESCHEMA" },

{ ECMD\_NEWFACET, "NEWFACET" },

{ ECMD\_VALIDATEXMLDATA, "VALIDATEXMLDATA" },

{ ECMD\_DOCOUTLINETOGGLE, "DOCOUTLINETOGGLE" },

{ ECMD\_VALIDATEHTMLDATA, "VALIDATEHTMLDATA" },

{ ECMD\_VIEWXMLSCHEMAOVERVIEW, "VIEWXMLSCHEMAOVERVIEW" },

{ ECMD\_SHOWDEFAULTVIEW, "SHOWDEFAULTVIEW" },

{ ECMD\_EXPAND\_CHILDREN, "EXPAND\_CHILDREN" },

{ ECMD\_COLLAPSE\_CHILDREN, "COLLAPSE\_CHILDREN" },

{ ECMD\_TOPDOWNLAYOUT, "TOPDOWNLAYOUT" },

{ ECMD\_LEFTRIGHTLAYOUT, "LEFTRIGHTLAYOUT" },

{ ECMD\_INSERTCELLRIGHT, "INSERTCELLRIGHT" },

{ ECMD\_EDITMASTER, "EDITMASTER" },

{ ECMD\_INSERTSNIPPET, "INSERTSNIPPET" },

{ ECMD\_FORMATANDVALIDATION, "FORMATANDVALIDATION" },

{ ECMD\_COLLAPSETAG, "COLLAPSETAG" },

{ ECMD\_SELECT\_TAG, "SELECT\_TAG" },

{ ECMD\_SELECT\_TAG\_CONTENT, "SELECT\_TAG\_CONTENT" },

{ ECMD\_CHECK\_ACCESSIBILITY, "CHECK\_ACCESSIBILITY" },

{ ECMD\_UNCOLLAPSETAG, "UNCOLLAPSETAG" },

{ ECMD\_GENERATEPAGERESOURCE, "GENERATEPAGERESOURCE" },

{ ECMD\_SHOWNONVISUALCONTROLS, "SHOWNONVISUALCONTROLS" },

{ ECMD\_RESIZECOLUMN, "RESIZECOLUMN" },

{ ECMD\_RESIZEROW, "RESIZEROW" },

{ ECMD\_MAKEABSOLUTE, "MAKEABSOLUTE" },

{ ECMD\_MAKERELATIVE, "MAKERELATIVE" },

{ ECMD\_MAKESTATIC, "MAKESTATIC" },

{ ECMD\_INSERTLAYER, "INSERTLAYER" },

{ ECMD\_UPDATEDESIGNVIEW, "UPDATEDESIGNVIEW" },

{ ECMD\_UPDATESOURCEVIEW, "UPDATESOURCEVIEW" },

{ ECMD\_INSERTCAPTION, "INSERTCAPTION" },

{ ECMD\_DELETECAPTION, "DELETECAPTION" },

{ ECMD\_MAKEPOSITIONNOTSET, "MAKEPOSITIONNOTSET" },

{ ECMD\_AUTOPOSITIONOPTIONS, "AUTOPOSITIONOPTIONS" },

{ ECMD\_EDITIMAGE, "EDITIMAGE" },

{ ECMD\_COMPILE, "COMPILE" },

{ ECMD\_PROJSETTINGS, "PROJSETTINGS" },

{ ECMD\_LINKONLY, "LINKONLY" },

{ ECMD\_REMOVE, "REMOVE" },

{ ECMD\_PROJSTARTDEBUG, "PROJSTARTDEBUG" },

{ ECMD\_PROJSTEPINTO, "PROJSTEPINTO" },

{ ECMD\_UPDATEMGDRES, "ECMD\_UPDATEMGDRES" },

{ ECMD\_UPDATEWEBREF, "UPDATEWEBREF" },

{ ECMD\_ADDRESOURCE, "ADDRESOURCE" },

{ ECMD\_WEBDEPLOY, "WEBDEPLOY" },

{ ECMD\_PROJTOOLORDER, "ECMD\_PROJTOOLORDER" },

{ ECMD\_PROJECTTOOLFILES, "ECMD\_PROJECTTOOLFILES" },

{ ECMD\_OTB\_PGO\_INSTRUMENT, "ECMD\_OTB\_PGO\_INSTRUMENT" },

{ ECMD\_OTB\_PGO\_OPT, "ECMD\_OTB\_PGO\_OPT" },

{ ECMD\_OTB\_PGO\_UPDATE, "ECMD\_OTB\_PGO\_UPDATE" },

{ ECMD\_OTB\_PGO\_RUNSCENARIO, "ECMD\_OTB\_PGO\_RUNSCENARIO" },

{ ECMD\_ADDHTMLPAGE, "ADDHTMLPAGE" },

{ ECMD\_ADDHTMLPAGECTX, "ADDHTMLPAGECTX" },

{ ECMD\_ADDMODULE, "ADDMODULE" },

{ ECMD\_ADDMODULECTX, "ADDMODULECTX" },

{ ECMD\_ADDWFCFORM, "ADDWFCFORM" },

{ ECMD\_ADDWEBFORM, "ADDWEBFORM" },

{ ECMD\_ADDMASTERPAGE, "ECMD\_ADDMASTERPAGE" },

{ ECMD\_ADDUSERCONTROL, "ADDUSERCONTROL" },

{ ECMD\_ADDCONTENTPAGE, "ECMD\_ADDCONTENTPAGE" },

{ ECMD\_ADDDHTMLPAGE, "ADDDHTMLPAGE" },

{ ECMD\_ADDIMAGEGENERATOR, "ADDIMAGEGENERATOR" },

{ ECMD\_ADDINHERWFCFORM, "ADDINHERWFCFORM" },

{ ECMD\_ADDINHERCONTROL, "ADDINHERCONTROL" },

{ ECMD\_ADDWEBUSERCONTROL, "ADDWEBUSERCONTROL" },

//        { ECMD\_BUILDANDBROWSE, "BUILDANDBROWSE" },

{ ECMD\_ADDTBXCOMPONENT, "ADDTBXCOMPONENT" },

{ ECMD\_ADDWEBSERVICE, "ADDWEBSERVICE" },

{ ECMD\_ADDSTYLESHEET, "ECMD\_ADDSTYLESHEET" },

{ ECMD\_SETBROWSELOCATION, "ECMD\_SETBROWSELOCATION" },

{ ECMD\_REFRESHFOLDER, "ECMD\_REFRESHFOLDER" },

{ ECMD\_SETBROWSELOCATIONCTX, "ECMD\_SETBROWSELOCATIONCTX" },

{ ECMD\_VIEWMARKUP, "ECMD\_VIEWMARKUP" },

{ ECMD\_NEXTMETHOD, "ECMD\_NEXTMETHOD" },

{ ECMD\_PREVMETHOD, "ECMD\_PREVMETHOD" },

{ ECMD\_RENAMESYMBOL, "ECMD\_RENAMESYMBOL" },

{ ECMD\_SHOWREFERENCES, "ECMD\_SHOWREFERENCES" },

{ ECMD\_CREATESNIPPET, "ECMD\_CREATESNIPPET" },

{ ECMD\_CREATEREPLACEMENT, "ECMD\_CREATEREPLACEMENT" },

{ ECMD\_INSERTCOMMENT, "ECMD\_INSERTCOMMENT" },

{ ECMD\_VIEWCOMPONENTDESIGNER, "VIEWCOMPONENTDESIGNER" },

{ ECMD\_GOTOTYPEDEF, "GOTOTYPEDEF" },

{ ECMD\_SHOWSNIPPETHIGHLIGHTING, "SHOWSNIPPETHIGHLIGHTING" },

{ ECMD\_HIDESNIPPETHIGHLIGHTING, "HIDESNIPPETHIGHLIGHTING" },

{ ECMD\_ADDVFPPAGE, "ADDVFPPAGE" },

{ ECMD\_SETBREAKPOINT, "SETBREAKPOINT" },

{ ECMD\_SHOWALLFILES, "SHOWALLFILES" },

{ ECMD\_ADDTOPROJECT, "ADDTOPROJECT" },

{ ECMD\_ADDBLANKNODE, "ADDBLANKNODE" },

{ ECMD\_ADDNODEFROMFILE, "ADDNODEFROMFILE" },

{ ECMD\_CHANGEURLFROMFILE, "CHANGEURLFROMFILE" },

{ ECMD\_EDITTOPIC, "EDITTOPIC" },

{ ECMD\_EDITTITLE, "EDITTITLE" },

{ ECMD\_MOVENODEUP, "MOVENODEUP" },

{ ECMD\_MOVENODEDOWN, "MOVENODEDOWN" },

{ ECMD\_MOVENODELEFT, "MOVENODELEFT" },

{ ECMD\_MOVENODERIGHT, "MOVENODERIGHT" },

{ ECMD\_ADDOUTPUT, "ADDOUTPUT" },

{ ECMD\_ADDFILE, "ADDFILE" },

{ ECMD\_MERGEMODULE, "MERGEMODULE" },

{ ECMD\_ADDCOMPONENTS, "ADDCOMPONENTS" },

{ ECMD\_LAUNCHINSTALLER, "LAUNCHINSTALLER" },

{ ECMD\_LAUNCHUNINSTALL, "LAUNCHUNINSTALL" },

{ ECMD\_LAUNCHORCA, "LAUNCHORCA" },

{ ECMD\_FILESYSTEMEDITOR, "FILESYSTEMEDITOR" },

{ ECMD\_REGISTRYEDITOR, "REGISTRYEDITOR" },

{ ECMD\_FILETYPESEDITOR, "FILETYPESEDITOR" },

{ ECMD\_USERINTERFACEEDITOR, "USERINTERFACEEDITOR" },

{ ECMD\_CUSTOMACTIONSEDITOR, "CUSTOMACTIONSEDITOR" },

{ ECMD\_LAUNCHCONDITIONSEDITOR, "LAUNCHCONDITIONSEDITOR" },

{ ECMD\_EDITOR, "EDITOR" },

{ ECMD\_EXCLUDE, "EXCLUDE" },

{ ECMD\_REFRESHDEPENDENCIES, "REFRESHDEPENDENCIES" },

{ ECMD\_VIEWOUTPUTS, "VIEWOUTPUTS" },

{ ECMD\_VIEWDEPENDENCIES, "VIEWDEPENDENCIES" },

{ ECMD\_VIEWFILTER, "VIEWFILTER" },

{ ECMD\_KEY, "KEY" },

{ ECMD\_STRING, "STRING" },

{ ECMD\_BINARY, "BINARY" },

{ ECMD\_DWORD, "DWORD" },

{ ECMD\_KEYSOLO, "KEYSOLO" },

{ ECMD\_IMPORT, "IMPORT" },

{ ECMD\_FOLDER, "FOLDER" },

{ ECMD\_PROJECTOUTPUT, "PROJECTOUTPUT" },

{ ECMD\_FILE, "FILE" },

{ ECMD\_ADDMERGEMODULES, "ADDMERGEMODULES" },

{ ECMD\_CREATESHORTCUT, "CREATESHORTCUT" },

{ ECMD\_LARGEICONS, "LARGEICONS" },

{ ECMD\_SMALLICONS, "SMALLICONS" },

{ ECMD\_LIST, "LIST" },

{ ECMD\_DETAILS, "DETAILS" },

{ ECMD\_ADDFILETYPE, "ADDFILETYPE" },

{ ECMD\_ADDACTION, "ADDACTION" },

{ ECMD\_SETASDEFAULT, "SETASDEFAULT" },

{ ECMD\_MOVEUP, "MOVEUP" },

{ ECMD\_MOVEDOWN, "MOVEDOWN" },

{ ECMD\_ADDDIALOG, "ADDDIALOG" },

{ ECMD\_IMPORTDIALOG, "IMPORTDIALOG" },

{ ECMD\_ADDFILESEARCH, "ADDFILESEARCH" },

{ ECMD\_ADDREGISTRYSEARCH, "ADDREGISTRYSEARCH" },

{ ECMD\_ADDCOMPONENTSEARCH, "ADDCOMPONENTSEARCH" },

{ ECMD\_ADDLAUNCHCONDITION, "ADDLAUNCHCONDITION" },

{ ECMD\_ADDCUSTOMACTION, "ADDCUSTOMACTION" },

{ ECMD\_OUTPUTS, "OUTPUTS" },

{ ECMD\_DEPENDENCIES, "DEPENDENCIES" },

{ ECMD\_FILTER, "FILTER" },

{ ECMD\_COMPONENTS, "COMPONENTS" },

{ ECMD\_ENVSTRING, "ENVSTRING" },

{ ECMD\_CREATEEMPTYSHORTCUT, "CREATEEMPTYSHORTCUT" },

{ ECMD\_ADDFILECONDITION, "ADDFILECONDITION" },

{ ECMD\_ADDREGISTRYCONDITION, "ADDREGISTRYCONDITION" },

{ ECMD\_ADDCOMPONENTCONDITION, "ADDCOMPONENTCONDITION" },

{ ECMD\_ADDURTCONDITION, "ADDURTCONDITION" },

{ ECMD\_ADDIISCONDITION, "ADDIISCONDITION" },

{ ECMD\_USERSAPPLICATIONDATAFOLDER, "USERSAPPLICATIONDATAFOLDER" },

{ ECMD\_SPECIALFOLDERBASE, "SPECIALFOLDERBASE" },

{ ECMD\_COMMONFILES64FOLDER, "COMMONFILES64FOLDER" },

{ ECMD\_COMMONFILESFOLDER, "COMMONFILESFOLDER" },

{ ECMD\_CUSTOMFOLDER, "CUSTOMFOLDER" },

{ ECMD\_USERSDESKTOP, "USERSDESKTOP" },

{ ECMD\_USERSFAVORITESFOLDER, "USERSFAVORITESFOLDER" },

{ ECMD\_FONTSFOLDER, "FONTSFOLDER" },

{ ECMD\_GLOBALASSEMBLYCACHEFOLDER, "GLOBALASSEMBLYCACHEFOLDER" },

{ ECMD\_MODULERETARGETABLEFOLDER, "MODULERETARGETABLEFOLDER" },

{ ECMD\_USERSPERSONALDATAFOLDER, "USERSPERSONALDATAFOLDER" },

{ ECMD\_PROGRAMFILES64FOLDER, "PROGRAMFILES64FOLDER" },

{ ECMD\_PROGRAMFILESFOLDER, "PROGRAMFILESFOLDER" },

{ ECMD\_USERSPROGRAMSMENU, "USERSPROGRAMSMENU" },

{ ECMD\_USERSSENDTOMENU, "USERSSENDTOMENU" },

{ ECMD\_SHAREDCOMPONENTSFOLDER, "SHAREDCOMPONENTSFOLDER" },

{ ECMD\_USERSSTARTMENU, "USERSSTARTMENU" },

{ ECMD\_USERSSTARTUPFOLDER, "USERSSTARTUPFOLDER" },

{ ECMD\_SYSTEM64FOLDER, "SYSTEM64FOLDER" },

{ ECMD\_SYSTEMFOLDER, "SYSTEMFOLDER" },

{ ECMD\_APPLICATIONFOLDER, "APPLICATIONFOLDER" },

{ ECMD\_USERSTEMPLATEFOLDER, "USERSTEMPLATEFOLDER" },

{ ECMD\_WEBCUSTOMFOLDER, "WEBCUSTOMFOLDER" },

{ ECMD\_WINDOWSFOLDER, "WINDOWSFOLDER" },

{ ECMD\_SPECIALFOLDERLAST, "SPECIALFOLDERLAST" },

{ ECMD\_EXPORTEVENTS, "EXPORTEVENTS" },

{ ECMD\_IMPORTEVENTS, "IMPORTEVENTS" },

{ ECMD\_VIEWEVENT, "VIEWEVENT" },

{ ECMD\_VIEWEVENTLIST, "VIEWEVENTLIST" },

{ ECMD\_VIEWCHART, "VIEWCHART" },

{ ECMD\_VIEWMACHINEDIAGRAM, "VIEWMACHINEDIAGRAM" },

{ ECMD\_VIEWPROCESSDIAGRAM, "VIEWPROCESSDIAGRAM" },

{ ECMD\_VIEWSOURCEDIAGRAM, "VIEWSOURCEDIAGRAM" },

{ ECMD\_VIEWSTRUCTUREDIAGRAM, "VIEWSTRUCTUREDIAGRAM" },

{ ECMD\_VIEWTIMELINE, "VIEWTIMELINE" },

{ ECMD\_VIEWSUMMARY, "VIEWSUMMARY" },

{ ECMD\_APPLYFILTER, "APPLYFILTER" },

{ ECMD\_CLEARFILTER, "CLEARFILTER" },

{ ECMD\_STARTRECORDING, "STARTRECORDING" },

{ ECMD\_STOPRECORDING, "STOPRECORDING" },

{ ECMD\_PAUSERECORDING, "PAUSERECORDING" },

{ ECMD\_ACTIVATEFILTER, "ACTIVATEFILTER" },

{ ECMD\_SHOWFIRSTEVENT, "SHOWFIRSTEVENT" },

{ ECMD\_SHOWPREVIOUSEVENT, "SHOWPREVIOUSEVENT" },

{ ECMD\_SHOWNEXTEVENT, "SHOWNEXTEVENT" },

{ ECMD\_SHOWLASTEVENT, "SHOWLASTEVENT" },

{ ECMD\_REPLAYEVENTS, "REPLAYEVENTS" },

{ ECMD\_STOPREPLAY, "STOPREPLAY" },

{ ECMD\_INCREASEPLAYBACKSPEED, "INCREASEPLAYBACKSPEED" },

{ ECMD\_DECREASEPLAYBACKSPEED, "DECREASEPLAYBACKSPEED" },

{ ECMD\_ADDMACHINE, "ADDMACHINE" },

{ ECMD\_ADDREMOVECOLUMNS, "ADDREMOVECOLUMNS" },

{ ECMD\_SORTCOLUMNS, "SORTCOLUMNS" },

{ ECMD\_SAVECOLUMNSETTINGS, "SAVECOLUMNSETTINGS" },

{ ECMD\_RESETCOLUMNSETTINGS, "RESETCOLUMNSETTINGS" },

{ ECMD\_SIZECOLUMNSTOFIT, "SIZECOLUMNSTOFIT" },

{ ECMD\_AUTOSELECT, "AUTOSELECT" },

{ ECMD\_AUTOFILTER, "AUTOFILTER" },

{ ECMD\_AUTOPLAYTRACK, "AUTOPLAYTRACK" },

{ ECMD\_GOTOEVENT, "GOTOEVENT" },

{ ECMD\_ZOOMTOFIT, "ZOOMTOFIT" },

{ ECMD\_ADDGRAPH, "ADDGRAPH" },

{ ECMD\_REMOVEGRAPH, "REMOVEGRAPH" },

{ ECMD\_CONNECTMACHINE, "CONNECTMACHINE" },

{ ECMD\_DISCONNECTMACHINE, "DISCONNECTMACHINE" },

{ ECMD\_EXPANDSELECTION, "EXPANDSELECTION" },

{ ECMD\_COLLAPSESELECTION, "COLLAPSESELECTION" },

{ ECMD\_ADDFILTER, "ADDFILTER" },

{ ECMD\_ADDPREDEFINED0, "ADDPREDEFINED0" },

{ ECMD\_ADDPREDEFINED1, "ADDPREDEFINED1" },

{ ECMD\_ADDPREDEFINED2, "ADDPREDEFINED2" },

{ ECMD\_ADDPREDEFINED3, "ADDPREDEFINED3" },

{ ECMD\_ADDPREDEFINED4, "ADDPREDEFINED4" },

{ ECMD\_ADDPREDEFINED5, "ADDPREDEFINED5" },

{ ECMD\_ADDPREDEFINED6, "ADDPREDEFINED6" },

{ ECMD\_ADDPREDEFINED7, "ADDPREDEFINED7" },

{ ECMD\_ADDPREDEFINED8, "ADDPREDEFINED8" },

{ ECMD\_TIMELINESIZETOFIT, "TIMELINESIZETOFIT" },

{ ECMD\_FIELDVIEW, "FIELDVIEW" },

{ ECMD\_SELECTEXPERT, "SELECTEXPERT" },

{ ECMD\_TOPNEXPERT, "TOPNEXPERT" },

{ ECMD\_SORTORDER, "SORTORDER" },

{ ECMD\_PROPPAGE, "PROPPAGE" },

{ ECMD\_HELP, "HELP" },

{ ECMD\_SAVEREPORT, "SAVEREPORT" },

{ ECMD\_INSERTSUMMARY, "INSERTSUMMARY" },

{ ECMD\_INSERTGROUP, "INSERTGROUP" },

{ ECMD\_INSERTSUBREPORT, "INSERTSUBREPORT" },

{ ECMD\_INSERTCHART, "INSERTCHART" },

{ ECMD\_INSERTPICTURE, "INSERTPICTURE" },

{ ECMD\_SETASSTARTPAGE, "SETASSTARTPAGE" },

{ ECMD\_RECALCULATELINKS, "RECALCULATELINKS" },

{ ECMD\_WEBPERMISSIONS, "WEBPERMISSIONS" },

{ ECMD\_COMPARETOMASTER, "COMPARETOMASTER" },

{ ECMD\_WORKOFFLINE, "WORKOFFLINE" },

{ ECMD\_SYNCHRONIZEFOLDER, "SYNCHRONIZEFOLDER" },

{ ECMD\_SYNCHRONIZEALLFOLDERS, "SYNCHRONIZEALLFOLDERS" },

{ ECMD\_COPYPROJECT, "COPYPROJECT" },

{ ECMD\_IMPORTFILEFROMWEB, "IMPORTFILEFROMWEB" },

{ ECMD\_INCLUDEINPROJECT, "INCLUDEINPROJECT" },

{ ECMD\_EXCLUDEFROMPROJECT, "EXCLUDEFROMPROJECT" },

{ ECMD\_BROKENLINKSREPORT, "BROKENLINKSREPORT" },

{ ECMD\_ADDPROJECTOUTPUTS, "ADDPROJECTOUTPUTS" },

{ ECMD\_ADDREFERENCE, "ADDREFERENCE" },

{ ECMD\_ADDWEBREFERENCE, "ADDWEBREFERENCE" },

{ ECMD\_ADDWEBREFERENCECTX, "ADDWEBREFERENCECTX" },

{ ECMD\_UPDATEWEBREFERENCE, "UPDATEWEBREFERENCE" },

{ ECMD\_RUNCUSTOMTOOL, "RUNCUSTOMTOOL" },

{ ECMD\_SETRUNTIMEVERSION, "SETRUNTIMEVERSION" },

//        { ECMD\_QUICKOBJECTSEARCH, "QUICKOBJECTSEARCH" },

{ ECMD\_VIEWREFINOBJECTBROWSER, "VIEWREFINOBJECTBROWSER" },

{ ECMD\_PUBLISH, "PUBLISH" },

{ ECMD\_PUBLISHCTX, "PUBLISHCTX" },

{ ECMD\_STARTOPTIONS, "STARTOPTIONS" },

{ ECMD\_ADDREFERENCECTX, "ADDREFERENCECTX" },

{ ECMD\_STARTOPTIONSCTX, "STARTOPTIONSCTX" },

{ ECMD\_DETACHLOCALDATAFILECTX, "DETACHLOCALDATAFILECTX" },

{ ECMD\_ADDSERVICEREFERENCE, "ADDSERVICEREFERENCE" },

{ ECMD\_ADDSERVICEREFERENCECTX, "ADDSERVICEREFERENCECTX" },

{ ECMD\_UPDATESERVICEREFERENCE, "UPDATESERVICEREFERENCE" },

{ ECMD\_CONFIGURESERVICEREFERENCE, "CONFIGURESERVICEREFERENCE" },

{ ECMD\_DRAG\_MOVE, "DRAG\_MOVE" },

{ ECMD\_DRAG\_COPY, "DRAG\_COPY" },

{ ECMD\_DRAG\_CANCEL, "DRAG\_CANCEL" },

{ ECMD\_TESTDIALOG, "TESTDIALOG" },

{ ECMD\_SPACEACROSS, "SPACEACROSS" },

{ ECMD\_SPACEDOWN, "SPACEDOWN" },

{ ECMD\_TOGGLEGRID, "TOGGLEGRID" },

{ ECMD\_TOGGLEGUIDES, "TOGGLEGUIDES" },

{ ECMD\_SIZETOTEXT, "SIZETOTEXT" },

{ ECMD\_CENTERVERT, "CENTERVERT" },

{ ECMD\_CENTERHORZ, "CENTERHORZ" },

{ ECMD\_FLIPDIALOG, "FLIPDIALOG" },

{ ECMD\_SETTABORDER, "SETTABORDER" },

{ ECMD\_BUTTONRIGHT, "BUTTONRIGHT" },

{ ECMD\_BUTTONBOTTOM, "BUTTONBOTTOM" },

{ ECMD\_AUTOLAYOUTGROW, "AUTOLAYOUTGROW" },

{ ECMD\_AUTOLAYOUTNORESIZE, "AUTOLAYOUTNORESIZE" },

{ ECMD\_AUTOLAYOUTOPTIMIZE, "AUTOLAYOUTOPTIMIZE" },

{ ECMD\_GUIDESETTINGS, "GUIDESETTINGS" },

{ ECMD\_RESOURCEINCLUDES, "RESOURCEINCLUDES" },

{ ECMD\_RESOURCESYMBOLS, "RESOURCESYMBOLS" },

{ ECMD\_OPENBINARY, "OPENBINARY" },

{ ECMD\_RESOURCEOPEN, "RESOURCEOPEN" },

{ ECMD\_RESOURCENEW, "RESOURCENEW" },

{ ECMD\_RESOURCENEWCOPY, "RESOURCENEWCOPY" },

{ ECMD\_INSERT, "INSERT" },

{ ECMD\_EXPORT, "EXPORT" },

{ ECMD\_CTLMOVELEFT, "CTLMOVELEFT" },

{ ECMD\_CTLMOVEDOWN, "CTLMOVEDOWN" },

{ ECMD\_CTLMOVERIGHT, "CTLMOVERIGHT" },

{ ECMD\_CTLMOVEUP, "CTLMOVEUP" },

{ ECMD\_CTLSIZEDOWN, "CTLSIZEDOWN" },

{ ECMD\_CTLSIZEUP, "CTLSIZEUP" },

{ ECMD\_CTLSIZELEFT, "CTLSIZELEFT" },

{ ECMD\_CTLSIZERIGHT, "CTLSIZERIGHT" },

{ ECMD\_NEWACCELERATOR, "NEWACCELERATOR" },

{ ECMD\_CAPTUREKEYSTROKE, "CAPTUREKEYSTROKE" },

{ ECMD\_INSERTACTIVEXCTL, "INSERTACTIVEXCTL" },

{ ECMD\_INVERTCOLORS, "INVERTCOLORS" },

{ ECMD\_FLIPHORIZONTAL, "FLIPHORIZONTAL" },

{ ECMD\_FLIPVERTICAL, "FLIPVERTICAL" },

{ ECMD\_ROTATE90, "ROTATE90" },

{ ECMD\_SHOWCOLORSWINDOW, "SHOWCOLORSWINDOW" },

{ ECMD\_NEWSTRING, "NEWSTRING" },

{ ECMD\_NEWINFOBLOCK, "NEWINFOBLOCK" },

{ ECMD\_DELETEINFOBLOCK, "DELETEINFOBLOCK" },

{ ECMD\_ADJUSTCOLORS, "ADJUSTCOLORS" },

{ ECMD\_LOADPALETTE, "LOADPALETTE" },

{ ECMD\_SAVEPALETTE, "SAVEPALETTE" },

{ ECMD\_CHECKMNEMONICS, "CHECKMNEMONICS" },

{ ECMD\_DRAWOPAQUE, "DRAWOPAQUE" },

{ ECMD\_TOOLBAREDITOR, "TOOLBAREDITOR" },

{ ECMD\_GRIDSETTINGS, "GRIDSETTINGS" },

{ ECMD\_NEWDEVICEIMAGE, "NEWDEVICEIMAGE" },

{ ECMD\_OPENDEVICEIMAGE, "OPENDEVICEIMAGE" },

{ ECMD\_DELETEDEVICEIMAGE, "DELETEDEVICEIMAGE" },

{ ECMD\_VIEWASPOPUP, "VIEWASPOPUP" },

{ ECMD\_CHECKMENUMNEMONICS, "CHECKMENUMNEMONICS" },

{ ECMD\_SHOWIMAGEGRID, "SHOWIMAGEGRID" },

{ ECMD\_SHOWTILEGRID, "SHOWTILEGRID" },

{ ECMD\_MAGNIFY, "MAGNIFY" },

{ cmdidResProps, "ResProps" },

{ ECMD\_IMPORTICONIMAGE, "IMPORTICONIMAGE" },

{ ECMD\_EXPORTICONIMAGE, "EXPORTICONIMAGE" },

{ ECMD\_OPENEXTERNALEDITOR, "OPENEXTERNALEDITOR" },

{ ECMD\_PICKRECTANGLE, "PICKRECTANGLE" },

{ ECMD\_PICKREGION, "PICKREGION" },

{ ECMD\_PICKCOLOR, "PICKCOLOR" },

{ ECMD\_ERASERTOOL, "ERASERTOOL" },

{ ECMD\_FILLTOOL, "FILLTOOL" },

{ ECMD\_PENCILTOOL, "PENCILTOOL" },

{ ECMD\_BRUSHTOOL, "BRUSHTOOL" },

{ ECMD\_AIRBRUSHTOOL, "AIRBRUSHTOOL" },

{ ECMD\_LINETOOL, "LINETOOL" },

{ ECMD\_CURVETOOL, "CURVETOOL" },

{ ECMD\_TEXTTOOL, "TEXTTOOL" },

{ ECMD\_RECTTOOL, "RECTTOOL" },

{ ECMD\_OUTLINERECTTOOL, "OUTLINERECTTOOL" },

{ ECMD\_FILLEDRECTTOOL, "FILLEDRECTTOOL" },

{ ECMD\_ROUNDRECTTOOL, "ROUNDRECTTOOL" },

{ ECMD\_OUTLINEROUNDRECTTOOL, "OUTLINEROUNDRECTTOOL" },

{ ECMD\_FILLEDROUNDRECTTOOL, "FILLEDROUNDRECTTOOL" },

{ ECMD\_ELLIPSETOOL, "ELLIPSETOOL" },

{ ECMD\_OUTLINEELLIPSETOOL, "OUTLINEELLIPSETOOL" },

{ ECMD\_FILLEDELLIPSETOOL, "FILLEDELLIPSETOOL" },

{ ECMD\_SETHOTSPOT, "SETHOTSPOT" },

{ ECMD\_ZOOMTOOL, "ZOOMTOOL" },

{ ECMD\_ZOOM1X, "ZOOM1X" },

{ ECMD\_ZOOM2X, "ZOOM2X" },

{ ECMD\_ZOOM6X, "ZOOM6X" },

{ ECMD\_ZOOM8X, "ZOOM8X" },

{ ECMD\_TRANSPARENTBCKGRND, "TRANSPARENTBCKGRND" },

{ ECMD\_OPAQUEBCKGRND, "OPAQUEBCKGRND" },

{ ECMD\_ERASERSMALL, "ERASERSMALL" },

{ ECMD\_ERASERMEDIUM, "ERASERMEDIUM" },

{ ECMD\_ERASERLARGE, "ERASERLARGE" },

{ ECMD\_ERASERLARGER, "ERASERLARGER" },

{ ECMD\_CIRCLELARGE, "CIRCLELARGE" },

{ ECMD\_CIRCLEMEDIUM, "CIRCLEMEDIUM" },

{ ECMD\_CIRCLESMALL, "CIRCLESMALL" },

{ ECMD\_SQUARELARGE, "SQUARELARGE" },

{ ECMD\_SQUAREMEDIUM, "SQUAREMEDIUM" },

{ ECMD\_SQUARESMALL, "SQUARESMALL" },

{ ECMD\_LEFTDIAGLARGE, "LEFTDIAGLARGE" },

{ ECMD\_LEFTDIAGMEDIUM, "LEFTDIAGMEDIUM" },

{ ECMD\_LEFTDIAGSMALL, "LEFTDIAGSMALL" },

{ ECMD\_RIGHTDIAGLARGE, "RIGHTDIAGLARGE" },

{ ECMD\_RIGHTDIAGMEDIUM, "RIGHTDIAGMEDIUM" },

{ ECMD\_RIGHTDIAGSMALL, "RIGHTDIAGSMALL" },

{ ECMD\_SPLASHSMALL, "SPLASHSMALL" },

{ ECMD\_SPLASHMEDIUM, "SPLASHMEDIUM" },

{ ECMD\_SPLASHLARGE, "SPLASHLARGE" },

{ ECMD\_LINESMALLER, "LINESMALLER" },

{ ECMD\_LINESMALL, "LINESMALL" },

{ ECMD\_LINEMEDIUM, "LINEMEDIUM" },

{ ECMD\_LINELARGE, "LINELARGE" },

{ ECMD\_LINELARGER, "LINELARGER" },

{ ECMD\_LARGERBRUSH, "LARGERBRUSH" },

{ ECMD\_LARGEBRUSH, "LARGEBRUSH" },

{ ECMD\_STDBRUSH, "STDBRUSH" },

{ ECMD\_SMALLBRUSH, "SMALLBRUSH" },

{ ECMD\_SMALLERBRUSH, "SMALLERBRUSH" },

{ ECMD\_ZOOMIN, "ZOOMIN" },

{ ECMD\_ZOOMOUT, "ZOOMOUT" },

{ ECMD\_PREVCOLOR, "PREVCOLOR" },

{ ECMD\_PREVECOLOR, "PREVECOLOR" },

{ ECMD\_NEXTCOLOR, "NEXTCOLOR" },

{ ECMD\_NEXTECOLOR, "NEXTECOLOR" },

{ ECMD\_IMG\_OPTIONS, "IMG\_OPTIONS" },

{ ECMD\_STARTWEBADMINTOOL, "STARTWEBADMINTOOL" },

{ ECMD\_NESTRELATEDFILES, "NESTRELATEDFILES" },

{ ECMD\_CANCELDRAG, "CANCELDRAG" },

{ ECMD\_DEFAULTACTION, "DEFAULTACTION" },

{ ECMD\_CTLMOVEUPGRID, "CTLMOVEUPGRID" },

{ ECMD\_CTLMOVEDOWNGRID, "CTLMOVEDOWNGRID" },

{ ECMD\_CTLMOVELEFTGRID, "CTLMOVELEFTGRID" },

{ ECMD\_CTLMOVERIGHTGRID, "CTLMOVERIGHTGRID" },

{ ECMD\_CTLSIZERIGHTGRID, "CTLSIZERIGHTGRID" },

{ ECMD\_CTLSIZEUPGRID, "CTLSIZEUPGRID" },

{ ECMD\_CTLSIZELEFTGRID, "CTLSIZELEFTGRID" },

{ ECMD\_CTLSIZEDOWNGRID, "CTLSIZEDOWNGRID" },

{ ECMD\_NEXTCTL, "NEXTCTL" },

{ ECMD\_PREVCTL, "PREVCTL" },

{ ECMD\_RENAME, "RENAME" },

{ ECMD\_EXTRACTMETHOD, "EXTRACTMETHOD" },

{ ECMD\_ENCAPSULATEFIELD, "ENCAPSULATEFIELD" },

{ ECMD\_EXTRACTINTERFACE, "EXTRACTINTERFACE" },

{ ECMD\_PROMOTELOCAL, "PROMOTELOCAL" },

{ ECMD\_REMOVEPARAMETERS, "REMOVEPARAMETERS" },

{ ECMD\_REORDERPARAMETERS, "REORDERPARAMETERS" },

{ ECMD\_GENERATEMETHODSTUB, "GENERATEMETHODSTUB" },

{ ECMD\_IMPLEMENTINTERFACEIMPLICIT, "IMPLEMENTINTERFACEIMPLICIT" },

{ ECMD\_IMPLEMENTINTERFACEEXPLICIT, "IMPLEMENTINTERFACEEXPLICIT" },

{ ECMD\_IMPLEMENTABSTRACTCLASS, "IMPLEMENTABSTRACTCLASS" },

{ ECMD\_SURROUNDWITH, "SURROUNDWITH" },

{ cmdidToggleWordWrapOW, "ToggleWordWrapOW" },

{ cmdidGotoNextLocationOW, "GotoNextLocationOW" },

{ cmdidGotoPrevLocationOW, "GotoPrevLocationOW" },

{ cmdidBuildOnlyProject, "BuildOnlyProject" },

{ cmdidRebuildOnlyProject, "RebuildOnlyProject" },

{ cmdidCleanOnlyProject, "CleanOnlyProject" },

{ cmdidSetBuildStartupsOnlyOnRun, "SetBuildStartupsOnlyOnRun" },

{ cmdidUnhideAll, "UnhideAll" },

{ cmdidHideFolder, "HideFolder" },

{ cmdidUnhideFolders, "UnhideFolders" },

{ cmdidCopyFullPathName, "CopyFullPathName" },

{ cmdidSaveFolderAsSolution, "SaveFolderAsSolution" },

{ cmdidManageUserSettings, "ManageUserSettings" },

{ cmdidNewSolutionFolder, "NewSolutionFolder" },

{ cmdidClearPaneOW, "ClearPaneOW" },

{ cmdidGotoErrorTagOW, "GotoErrorTagOW" },

{ cmdidGotoNextErrorTagOW, "GotoNextErrorTagOW" },

{ cmdidGotoPrevErrorTagOW, "GotoPrevErrorTagOW" },

{ cmdidClearPaneFR1, "ClearPaneFR1" },

{ cmdidGotoErrorTagFR1, "GotoErrorTagFR1" },

{ cmdidGotoNextErrorTagFR1, "GotoNextErrorTagFR1" },

{ cmdidGotoPrevErrorTagFR1, "GotoPrevErrorTagFR1" },

{ cmdidClearPaneFR2, "ClearPaneFR2" },

{ cmdidGotoErrorTagFR2, "GotoErrorTagFR2" },

{ cmdidGotoNextErrorTagFR2, "GotoNextErrorTagFR2" },

{ cmdidGotoPrevErrorTagFR2, "GotoPrevErrorTagFR2" },

{ cmdidOutputPaneCombo, "OutputPaneCombo" },

{ cmdidOutputPaneComboList, "OutputPaneComboList" },

{ cmdidDisableDockingChanges, "DisableDockingChanges" },

{ cmdidToggleFloat, "ToggleFloat" },

{ cmdidResetLayout, "ResetLayout" },

{ cmdidNewSolutionFolderBar, "NewSolutionFolderBar" },

{ cmdidDataShortcut, "DataShortcut" },

{ cmdidNextToolWindow, "NextToolWindow" },

{ cmdidPrevToolWindow, "PrevToolWindow" },

{ cmdidBrowseToFileInExplorer, "BrowseToFileInExplorer" },

{ cmdidShowEzMDIFileMenu, "ShowEzMDIFileMenu" },

{ cmdidPrevToolWindowNav, "PrevToolWindowNav" },

{ cmdidStaticAnalysisOnlyProject, "StaticAnalysisOnlyProject" },

{ ECMD\_RUNFXCOPSEL, "ECMD\_RUNFXCOPSEL" },

{ cmdidCloseAllButThis, "CloseAllButThis" },

{ cmdidCVShowInheritedMembers, "CVShowInheritedMembers" },

{ cmdidCVShowBaseTypes, "CVShowBaseTypes" },

{ cmdidCVShowDerivedTypes, "CVShowDerivedTypes" },

{ cmdidCVShowHidden, "CVShowHidden" },

{ cmdidCVBack, "CVBack" },

{ cmdidCVForward, "CVForward" },

{ cmdidCVSearchCombo, "CVSearchCombo" },

{ cmdidCVSearch, "CVSearch" },

{ cmdidCVSortObjectsAlpha, "CVSortObjectsAlpha" },

{ cmdidCVSortObjectsType, "CVSortObjectsType" },

{ cmdidCVSortObjectsAccess, "CVSortObjectsAccess" },

{ cmdidCVGroupObjectsType, "CVGroupObjectsType" },

{ cmdidCVSortMembersAlpha, "CVSortMembersAlpha" },

{ cmdidCVSortMembersType, "CVSortMembersType" },

{ cmdidCVSortMembersAccess, "CVSortMembersAccess" },

{ cmdidCVTypeBrowserSettings, "CVTypeBrowserSettings" },

{ cmdidCVViewMembersAsImplementor, "CVViewMembersAsImplementor" },

{ cmdidCVViewMembersAsSubclass, "CVViewMembersAsSubclass" },

{ cmdidCVViewMembersAsUser, "CVViewMembersAsUser" },

{ cmdidCVReserved1, "CVReserved1" },

{ cmdidCVReserved2, "CVReserved2" },

{ cmdidCVShowProjectReferences, "CVShowProjectReferences" },

{ cmdidCVGroupMembersType, "CVGroupMembersType" },

{ cmdidCVClearSearch, "CVClearSearch" },

{ cmdidCVFilterToType, "CVFilterToType" },

{ cmdidCVSortByBestMatch, "CVSortByBestMatch" },

{ cmdidCVSearchMRUList, "CVSearchMRUList" },

{ cmdidCVViewOtherMembers, "CVViewOtherMembers" },

{ cmdidCVSearchCmd, "CVSearchCmd" },

{ cmdidCVGoToSearchCmd, "CVGoToSearchCmd" },

{ cmdidControlGallery, "ControlGallery" },

{ cmdidOBShowInheritedMembers, "OBShowInheritedMembers" },

{ cmdidOBShowBaseTypes, "OBShowBaseTypes" },

{ cmdidOBShowDerivedTypes, "OBShowDerivedTypes" },

{ cmdidOBShowHidden, "OBShowHidden" },

{ cmdidOBBack, "OBBack" },

{ cmdidOBForward, "OBForward" },

{ cmdidOBSearchCombo, "OBSearchCombo" },

{ cmdidOBSearch, "OBSearch" },

{ cmdidOBSortObjectsAlpha, "OBSortObjectsAlpha" },

{ cmdidOBSortObjectsType, "OBSortObjectsType" },

{ cmdidOBSortObjectsAccess, "OBSortObjectsAccess" },

{ cmdidOBGroupObjectsType, "OBGroupObjectsType" },

{ cmdidOBSortMembersAlpha, "OBSortMembersAlpha" },

{ cmdidOBSortMembersType, "OBSortMembersType" },

{ cmdidOBSortMembersAccess, "OBSortMembersAccess" },

{ cmdidOBTypeBrowserSettings, "OBTypeBrowserSettings" },

{ cmdidOBViewMembersAsImplementor, "OBViewMembersAsImplementor" },

{ cmdidOBViewMembersAsSubclass, "OBViewMembersAsSubclass" },

{ cmdidOBViewMembersAsUser, "OBViewMembersAsUser" },

{ cmdidOBNamespacesView, "OBNamespacesView" },

{ cmdidOBContainersView, "OBContainersView" },

{ cmdidOBReserved1, "OBReserved1" },

{ cmdidOBGroupMembersType, "OBGroupMembersType" },

{ cmdidOBClearSearch, "OBClearSearch" },

{ cmdidOBFilterToType, "OBFilterToType" },

{ cmdidOBSortByBestMatch, "OBSortByBestMatch" },

{ cmdidOBSearchMRUList, "OBSearchMRUList" },

{ cmdidOBViewOtherMembers, "OBViewOtherMembers" },

{ cmdidOBSearchCmd, "OBSearchCmd" },

{ cmdidOBGoToSearchCmd, "OBGoToSearchCmd" },

{ cmdidOBShowExtensionMembers, "OBShowExtensionMembers" },

{ cmdidFullScreen2, "FullScreen2" },

{ cmdidFSRSortObjectsAlpha, "FSRSortObjectsAlpha" },

{ cmdidFSRSortByBestMatch, "FSRSortByBestMatch" },

{ cmdidNavigateBack, "NavigateBack" },

{ cmdidNavigateForward, "NavigateForward" },

{ ECMD\_CORRECTION\_1, "ECMD\_CORRECTION\_1" },

{ ECMD\_CORRECTION\_2, "ECMD\_CORRECTION\_2" },

{ ECMD\_CORRECTION\_3, "ECMD\_CORRECTION\_3" },

{ ECMD\_CORRECTION\_4, "ECMD\_CORRECTION\_4" },

{ ECMD\_CORRECTION\_5, "ECMD\_CORRECTION\_5" },

{ ECMD\_CORRECTION\_6, "ECMD\_CORRECTION\_6" },

{ ECMD\_CORRECTION\_7, "ECMD\_CORRECTION\_7" },

{ ECMD\_CORRECTION\_8, "ECMD\_CORRECTION\_8" },

{ ECMD\_CORRECTION\_9, "ECMD\_CORRECTION\_9" },

{ ECMD\_CORRECTION\_10, "ECMD\_CORRECTION\_10" },

{ cmdidOBAddReference, "OBAddReference" },

{ cmdidFindReferences, "FindReferences" },

{ cmdidCodeDefView, "CodeDefView" },

{ cmdidCodeDefViewGoToPrev, "CodeDefViewGoToPrev" },

{ cmdidCodeDefViewGoToNext, "CodeDefViewGoToNext" },

{ cmdidCodeDefViewEditDefinition, "CodeDefViewEditDefinition" },

{ cmdidCodeDefViewChooseEncoding, "CodeDefViewChooseEncoding" },

{ cmdidViewInClassDiagram, "ViewInClassDiagram" },

{ ECMD\_ADDDBTABLE, "ECMD\_ADDDBTABLE" },

{ ECMD\_ADDDATATABLE, "ECMD\_ADDDATATABLE" },

{ ECMD\_ADDFUNCTION, "ECMD\_ADDFUNCTION" },

{ ECMD\_ADDRELATION, "ECMD\_ADDRELATION" },

{ ECMD\_ADDKEY, "ECMD\_ADDKEY" },

{ ECMD\_ADDCOLUMN, "ECMD\_ADDCOLUMN" },

{ ECMD\_CONVERT\_DBTABLE, "ECMD\_CONVERT\_DBTABLE" },

{ ECMD\_CONVERT\_DATATABLE, "ECMD\_CONVERT\_DATATABLE" },

{ ECMD\_GENERATE\_DATABASE, "ECMD\_GENERATE\_DATABASE" },

{ ECMD\_CONFIGURE\_CONNECTIONS, "ECMD\_CONFIGURE\_CONNECTIONS" },

{ ECMD\_IMPORT\_XMLSCHEMA, "ECMD\_IMPORT\_XMLSCHEMA" },

{ ECMD\_SYNC\_WITH\_DATABASE, "ECMD\_SYNC\_WITH\_DATABASE" },

{ ECMD\_CONFIGURE, "ECMD\_CONFIGURE" },

{ ECMD\_CREATE\_DATAFORM, "ECMD\_CREATE\_DATAFORM" },

{ ECMD\_CREATE\_ENUM, "ECMD\_CREATE\_ENUM" },

{ ECMD\_INSERT\_FUNCTION, "ECMD\_INSERT\_FUNCTION" },

{ ECMD\_EDIT\_FUNCTION, "ECMD\_EDIT\_FUNCTION" },

{ ECMD\_SET\_PRIMARY\_KEY, "ECMD\_SET\_PRIMARY\_KEY" },

{ ECMD\_INSERT\_COLUMN, "ECMD\_INSERT\_COLUMN" },

{ ECMD\_AUTO\_SIZE, "ECMD\_AUTO\_SIZE" },

{ ECMD\_SHOW\_RELATION\_LABELS, "ECMD\_SHOW\_RELATION\_LABELS" },

{ cmdid\_VSD\_GenerateDataSet, "VSDGenerateDataSet" },

{ cmdid\_VSD\_Preview, "VSDPreview" },

{ cmdid\_VSD\_ConfigureAdapter, "VSDConfigureAdapter" },

{ cmdid\_VSD\_ViewDatasetSchema, "VSDViewDatasetSchema" },

{ cmdid\_VSD\_DatasetProperties, "VSDDatasetProperties" },

{ cmdid\_VSD\_ParameterizeForm, "VSDParameterizeForm" },

{ cmdid\_VSD\_AddChildForm, "VSDAddChildForm" },

{ ECMD\_EDITCONSTRAINT, "ECMD\_EDITCONSTRAINT" },

{ ECMD\_DELETECONSTRAINT, "ECMD\_DELETECONSTRAINT" },

{ ECMD\_EDITDATARELATION, "ECMD\_EDITDATARELATION" },

{ cmdidCloseProject, "CloseProject" },

{ cmdidReloadCommandBars, "ReloadCommandBars" },

{ cmdidSolutionPlatform, "SolutionPlatform" },

{ cmdidSolutionPlatformGetList, "SolutionPlatformGetList" },

{ ECMD\_DATAACCESSOR, "ECMD\_DATAACCESSOR" },

{ ECMD\_ADD\_DATAACCESSOR, "ECMD\_ADD\_DATAACCESSOR" },

{ ECMD\_QUERY, "ECMD\_QUERY" },

{ ECMD\_ADD\_QUERY, "ECMD\_ADD\_QUERY" },

{ ECMD\_PUBLISHSELECTION, "ECMD\_PUBLISHSELECTION" },

{ ECMD\_PUBLISHSLNCTX, "ECMD\_PUBLISHSLNCTX" },

{ cmdidCallBrowserShowCallsTo, "CallBrowserShowCallsTo" },

{ cmdidCallBrowserShowCallsFrom, "CallBrowserShowCallsFrom" },

{ cmdidCallBrowserShowNewCallsTo, "CallBrowserShowNewCallsTo" },

{ cmdidCallBrowserShowNewCallsFrom, "CallBrowserShowNewCallsFrom" },

{ cmdidCallBrowser1ShowCallsTo, "CallBrowser1ShowCallsTo" },

{ cmdidCallBrowser2ShowCallsTo, "CallBrowser2ShowCallsTo" },

{ cmdidCallBrowser3ShowCallsTo, "CallBrowser3ShowCallsTo" },

{ cmdidCallBrowser4ShowCallsTo, "CallBrowser4ShowCallsTo" },

{ cmdidCallBrowser5ShowCallsTo, "CallBrowser5ShowCallsTo" },

{ cmdidCallBrowser6ShowCallsTo, "CallBrowser6ShowCallsTo" },

{ cmdidCallBrowser7ShowCallsTo, "CallBrowser7ShowCallsTo" },

{ cmdidCallBrowser8ShowCallsTo, "CallBrowser8ShowCallsTo" },

{ cmdidCallBrowser9ShowCallsTo, "CallBrowser9ShowCallsTo" },

{ cmdidCallBrowser10ShowCallsTo, "CallBrowser10ShowCallsTo" },

{ cmdidCallBrowser11ShowCallsTo, "CallBrowser11ShowCallsTo" },

{ cmdidCallBrowser12ShowCallsTo, "CallBrowser12ShowCallsTo" },

{ cmdidCallBrowser13ShowCallsTo, "CallBrowser13ShowCallsTo" },

{ cmdidCallBrowser14ShowCallsTo, "CallBrowser14ShowCallsTo" },

{ cmdidCallBrowser15ShowCallsTo, "CallBrowser15ShowCallsTo" },

{ cmdidCallBrowser16ShowCallsTo, "CallBrowser16ShowCallsTo" },

{ cmdidCallBrowser1ShowCallsFrom, "CallBrowser1ShowCallsFrom" },

{ cmdidCallBrowser2ShowCallsFrom, "CallBrowser2ShowCallsFrom" },

{ cmdidCallBrowser3ShowCallsFrom, "CallBrowser3ShowCallsFrom" },

{ cmdidCallBrowser4ShowCallsFrom, "CallBrowser4ShowCallsFrom" },

{ cmdidCallBrowser5ShowCallsFrom, "CallBrowser5ShowCallsFrom" },

{ cmdidCallBrowser6ShowCallsFrom, "CallBrowser6ShowCallsFrom" },

{ cmdidCallBrowser7ShowCallsFrom, "CallBrowser7ShowCallsFrom" },

{ cmdidCallBrowser8ShowCallsFrom, "CallBrowser8ShowCallsFrom" },

{ cmdidCallBrowser9ShowCallsFrom, "CallBrowser9ShowCallsFrom" },

{ cmdidCallBrowser10ShowCallsFrom, "CallBrowser10ShowCallsFrom" },

{ cmdidCallBrowser11ShowCallsFrom, "CallBrowser11ShowCallsFrom" },

{ cmdidCallBrowser12ShowCallsFrom, "CallBrowser12ShowCallsFrom" },

{ cmdidCallBrowser13ShowCallsFrom, "CallBrowser13ShowCallsFrom" },

{ cmdidCallBrowser14ShowCallsFrom, "CallBrowser14ShowCallsFrom" },

{ cmdidCallBrowser15ShowCallsFrom, "CallBrowser15ShowCallsFrom" },

{ cmdidCallBrowser16ShowCallsFrom, "CallBrowser16ShowCallsFrom" },

{ cmdidCallBrowser1ShowFullNames, "CallBrowser1ShowFullNames" },

{ cmdidCallBrowser2ShowFullNames, "CallBrowser2ShowFullNames" },

{ cmdidCallBrowser3ShowFullNames, "CallBrowser3ShowFullNames" },

{ cmdidCallBrowser4ShowFullNames, "CallBrowser4ShowFullNames" },

{ cmdidCallBrowser5ShowFullNames, "CallBrowser5ShowFullNames" },

{ cmdidCallBrowser6ShowFullNames, "CallBrowser6ShowFullNames" },

{ cmdidCallBrowser7ShowFullNames, "CallBrowser7ShowFullNames" },

{ cmdidCallBrowser8ShowFullNames, "CallBrowser8ShowFullNames" },

{ cmdidCallBrowser9ShowFullNames, "CallBrowser9ShowFullNames" },

{ cmdidCallBrowser10ShowFullNames, "CallBrowser10ShowFullNames" },

{ cmdidCallBrowser11ShowFullNames, "CallBrowser11ShowFullNames" },

{ cmdidCallBrowser12ShowFullNames, "CallBrowser12ShowFullNames" },

{ cmdidCallBrowser13ShowFullNames, "CallBrowser13ShowFullNames" },

{ cmdidCallBrowser14ShowFullNames, "CallBrowser14ShowFullNames" },

{ cmdidCallBrowser15ShowFullNames, "CallBrowser15ShowFullNames" },

{ cmdidCallBrowser16ShowFullNames, "CallBrowser16ShowFullNames" },

{ cmdidCallBrowser1Settings, "CallBrowser1Settings" },

{ cmdidCallBrowser2Settings, "CallBrowser2Settings" },

{ cmdidCallBrowser3Settings, "CallBrowser3Settings" },

{ cmdidCallBrowser4Settings, "CallBrowser4Settings" },

{ cmdidCallBrowser5Settings, "CallBrowser5Settings" },

{ cmdidCallBrowser6Settings, "CallBrowser6Settings" },

{ cmdidCallBrowser7Settings, "CallBrowser7Settings" },

{ cmdidCallBrowser8Settings, "CallBrowser8Settings" },

{ cmdidCallBrowser9Settings, "CallBrowser9Settings" },

{ cmdidCallBrowser10Settings, "CallBrowser10Settings" },

{ cmdidCallBrowser11Settings, "CallBrowser11Settings" },

{ cmdidCallBrowser12Settings, "CallBrowser12Settings" },

{ cmdidCallBrowser13Settings, "CallBrowser13Settings" },

{ cmdidCallBrowser14Settings, "CallBrowser14Settings" },

{ cmdidCallBrowser15Settings, "CallBrowser15Settings" },

{ cmdidCallBrowser16Settings, "CallBrowser16Settings" },

{ cmdidCallBrowser1SortAlpha, "CallBrowser1SortAlpha" },

{ cmdidCallBrowser2SortAlpha, "CallBrowser2SortAlpha" },

{ cmdidCallBrowser3SortAlpha, "CallBrowser3SortAlpha" },

{ cmdidCallBrowser4SortAlpha, "CallBrowser4SortAlpha" },

{ cmdidCallBrowser5SortAlpha, "CallBrowser5SortAlpha" },

{ cmdidCallBrowser6SortAlpha, "CallBrowser6SortAlpha" },

{ cmdidCallBrowser7SortAlpha, "CallBrowser7SortAlpha" },

{ cmdidCallBrowser8SortAlpha, "CallBrowser8SortAlpha" },

{ cmdidCallBrowser9SortAlpha, "CallBrowser9SortAlpha" },

{ cmdidCallBrowser10SortAlpha, "CallBrowser10SortAlpha" },

{ cmdidCallBrowser11SortAlpha, "CallBrowser11SortAlpha" },

{ cmdidCallBrowser12SortAlpha, "CallBrowser12SortAlpha" },

{ cmdidCallBrowser13SortAlpha, "CallBrowser13SortAlpha" },

{ cmdidCallBrowser14SortAlpha, "CallBrowser14SortAlpha" },

{ cmdidCallBrowser15SortAlpha, "CallBrowser15SortAlpha" },

{ cmdidCallBrowser16SortAlpha, "CallBrowser16SortAlpha" },

{ cmdidCallBrowser1SortAccess, "CallBrowser1SortAccess" },

{ cmdidCallBrowser2SortAccess, "CallBrowser2SortAccess" },

{ cmdidCallBrowser3SortAccess, "CallBrowser3SortAccess" },

{ cmdidCallBrowser4SortAccess, "CallBrowser4SortAccess" },

{ cmdidCallBrowser5SortAccess, "CallBrowser5SortAccess" },

{ cmdidCallBrowser6SortAccess, "CallBrowser6SortAccess" },

{ cmdidCallBrowser7SortAccess, "CallBrowser7SortAccess" },

{ cmdidCallBrowser8SortAccess, "CallBrowser8SortAccess" },

{ cmdidCallBrowser9SortAccess, "CallBrowser9SortAccess" },

{ cmdidCallBrowser10SortAccess, "CallBrowser10SortAccess" },

{ cmdidCallBrowser11SortAccess, "CallBrowser11SortAccess" },

{ cmdidCallBrowser12SortAccess, "CallBrowser12SortAccess" },

{ cmdidCallBrowser13SortAccess, "CallBrowser13SortAccess" },

{ cmdidCallBrowser14SortAccess, "CallBrowser14SortAccess" },

{ cmdidCallBrowser15SortAccess, "CallBrowser15SortAccess" },

{ cmdidCallBrowser16SortAccess, "CallBrowser16SortAccess" },

//        { cmdidShowCallBrowser, "ShowCallBrowser" },

{ cmdidCallBrowser1, "CallBrowser1" },

{ cmdidCallBrowser2, "CallBrowser2" },

{ cmdidCallBrowser3, "CallBrowser3" },

{ cmdidCallBrowser4, "CallBrowser4" },

{ cmdidCallBrowser5, "CallBrowser5" },

{ cmdidCallBrowser6, "CallBrowser6" },

{ cmdidCallBrowser7, "CallBrowser7" },

{ cmdidCallBrowser8, "CallBrowser8" },

{ cmdidCallBrowser9, "CallBrowser9" },

{ cmdidCallBrowser10, "CallBrowser10" },

{ cmdidCallBrowser11, "CallBrowser11" },

{ cmdidCallBrowser12, "CallBrowser12" },

{ cmdidCallBrowser13, "CallBrowser13" },

{ cmdidCallBrowser14, "CallBrowser14" },

{ cmdidCallBrowser15, "CallBrowser15" },

{ cmdidCallBrowser16, "CallBrowser16" },

{ cmdidCallBrowser17, "CallBrowser17" },

{ cmdidGlobalUndo, "GlobalUndo" },

{ cmdidGlobalRedo, "GlobalRedo" },

{ cmdidCallBrowserShowCallsToCmd, "CallBrowserShowCallsToCmd" },

{ cmdidCallBrowserShowCallsFromCmd, "CallBrowserShowCallsFromCmd" },

{ cmdidCallBrowserShowNewCallsToCmd, "CallBrowserShowNewCallsToCmd" },

{ cmdidCallBrowserShowNewCallsFromCmd, "CallBrowserShowNewCallsFromCmd" },

{ cmdidCallBrowser1Search, "CallBrowser1Search" },

{ cmdidCallBrowser2Search, "CallBrowser2Search" },

{ cmdidCallBrowser3Search, "CallBrowser3Search" },

{ cmdidCallBrowser4Search, "CallBrowser4Search" },

{ cmdidCallBrowser5Search, "CallBrowser5Search" },

{ cmdidCallBrowser6Search, "CallBrowser6Search" },

{ cmdidCallBrowser7Search, "CallBrowser7Search" },

{ cmdidCallBrowser8Search, "CallBrowser8Search" },

{ cmdidCallBrowser9Search, "CallBrowser9Search" },

{ cmdidCallBrowser10Search, "CallBrowser10Search" },

{ cmdidCallBrowser11Search, "CallBrowser11Search" },

{ cmdidCallBrowser12Search, "CallBrowser12Search" },

{ cmdidCallBrowser13Search, "CallBrowser13Search" },

{ cmdidCallBrowser14Search, "CallBrowser14Search" },

{ cmdidCallBrowser15Search, "CallBrowser15Search" },

{ cmdidCallBrowser16Search, "CallBrowser16Search" },

{ cmdidCallBrowser1Refresh, "CallBrowser1Refresh" },

{ cmdidCallBrowser2Refresh, "CallBrowser2Refresh" },

{ cmdidCallBrowser3Refresh, "CallBrowser3Refresh" },

{ cmdidCallBrowser4Refresh, "CallBrowser4Refresh" },

{ cmdidCallBrowser5Refresh, "CallBrowser5Refresh" },

{ cmdidCallBrowser6Refresh, "CallBrowser6Refresh" },

{ cmdidCallBrowser7Refresh, "CallBrowser7Refresh" },

{ cmdidCallBrowser8Refresh, "CallBrowser8Refresh" },

{ cmdidCallBrowser9Refresh, "CallBrowser9Refresh" },

{ cmdidCallBrowser10Refresh, "CallBrowser10Refresh" },

{ cmdidCallBrowser11Refresh, "CallBrowser11Refresh" },

{ cmdidCallBrowser12Refresh, "CallBrowser12Refresh" },

{ cmdidCallBrowser13Refresh, "CallBrowser13Refresh" },

{ cmdidCallBrowser14Refresh, "CallBrowser14Refresh" },

{ cmdidCallBrowser15Refresh, "CallBrowser15Refresh" },

{ cmdidCallBrowser16Refresh, "CallBrowser16Refresh" },

{ cmdidCallBrowser1SearchCombo, "CallBrowser1SearchCombo" },

{ cmdidCallBrowser2SearchCombo, "CallBrowser2SearchCombo" },

{ cmdidCallBrowser3SearchCombo, "CallBrowser3SearchCombo" },

{ cmdidCallBrowser4SearchCombo, "CallBrowser4SearchCombo" },

{ cmdidCallBrowser5SearchCombo, "CallBrowser5SearchCombo" },

{ cmdidCallBrowser6SearchCombo, "CallBrowser6SearchCombo" },

{ cmdidCallBrowser7SearchCombo, "CallBrowser7SearchCombo" },

{ cmdidCallBrowser8SearchCombo, "CallBrowser8SearchCombo" },

{ cmdidCallBrowser9SearchCombo, "CallBrowser9SearchCombo" },

{ cmdidCallBrowser10SearchCombo, "CallBrowser10SearchCombo" },

{ cmdidCallBrowser11SearchCombo, "CallBrowser11SearchCombo" },

{ cmdidCallBrowser12SearchCombo, "CallBrowser12SearchCombo" },

{ cmdidCallBrowser13SearchCombo, "CallBrowser13SearchCombo" },

{ cmdidCallBrowser14SearchCombo, "CallBrowser14SearchCombo" },

{ cmdidCallBrowser15SearchCombo, "CallBrowser15SearchCombo" },

{ cmdidCallBrowser16SearchCombo, "CallBrowser16SearchCombo" },

{ cmdidTaskListProviderCombo, "TaskListProviderCombo" },

{ cmdidTaskListProviderComboList, "TaskListProviderComboList" },

{ cmdidCreateUserTask, "CreateUserTask" },

{ cmdidErrorListShowErrors, "ErrorListShowErrors" },

{ cmdidErrorListShowWarnings, "ErrorListShowWarnings" },

{ cmdidErrorListShowMessages, "ErrorListShowMessages" },

{ cmdidRegistration, "Registration" },

{ cmdidCallBrowser1SearchComboList, "CallBrowser1SearchComboList" },

{ cmdidCallBrowser2SearchComboList, "CallBrowser2SearchComboList" },

{ cmdidCallBrowser3SearchComboList, "CallBrowser3SearchComboList" },

{ cmdidCallBrowser4SearchComboList, "CallBrowser4SearchComboList" },

{ cmdidCallBrowser5SearchComboList, "CallBrowser5SearchComboList" },

{ cmdidCallBrowser6SearchComboList, "CallBrowser6SearchComboList" },

{ cmdidCallBrowser7SearchComboList, "CallBrowser7SearchComboList" },

{ cmdidCallBrowser8SearchComboList, "CallBrowser8SearchComboList" },

{ cmdidCallBrowser9SearchComboList, "CallBrowser9SearchComboList" },

{ cmdidCallBrowser10SearchComboList, "CallBrowser10SearchComboList" },

{ cmdidCallBrowser11SearchComboList, "CallBrowser11SearchComboList" },

{ cmdidCallBrowser12SearchComboList, "CallBrowser12SearchComboList" },

{ cmdidCallBrowser13SearchComboList, "CallBrowser13SearchComboList" },

{ cmdidCallBrowser14SearchComboList, "CallBrowser14SearchComboList" },

{ cmdidCallBrowser15SearchComboList, "CallBrowser15SearchComboList" },

{ cmdidCallBrowser16SearchComboList, "CallBrowser16SearchComboList" },

{ cmdidSnippetProp, "SnippetProp" },

{ cmdidSnippetRef, "SnippetRef" },

{ cmdidSnippetRepl, "SnippetRepl" },

{ cmdidStartPage, "StartPage" },

{ cmdidEditorLineFirstColumn, "EditorLineFirstColumn" },

{ cmdidEditorLineFirstColumnExtend, "EditorLineFirstColumnExtend" },

{ cmdid\_SE\_ServerExplorer, "SEServerExplorer" },

{ cmdid\_SE\_DataExplorer, "SEDataExplorer" },

{ ECMD\_VALIDATION\_TARGET, "ECMD\_VALIDATION\_TARGET" },

{ ECMD\_VALIDATION\_TARGET\_GET\_LIST, "ECMD\_VALIDATION\_TARGET\_GET\_LIST" },

{ ECMD\_CSS\_TARGET, "ECMD\_CSS\_TARGET" },

{ ECMD\_CSS\_TARGET\_GET\_LIST, "ECMD\_CSS\_TARGET\_GET\_LIST" },

{ icmdDesign, "Design" },

{ icmdDesignOn, "DesignOn" },

{ icmdSEDesign, "SEDesign" },

{ icmdNewDiagram, "NewDiagram" },

{ icmdNewTable, "NewTable" },

{ icmdNewDBItem, "NewDBItem" },

{ icmdNewTrigger, "NewTrigger" },

{ icmdDebug, "Debug" },

{ icmdNewProcedure, "NewProcedure" },

{ icmdNewQuery, "NewQuery" },

{ icmdRefreshLocal, "RefreshLocal" },

{ icmdDbAddDataConnection, "DbAddDataConnection" },

{ icmdDBDefDBRef, "DBDefDBRef" },

{ icmdRunCmd, "RunCmd" },

{ icmdRunOn, "RunOn" },

{ icmdidNewDBRef, "NewDBRef" },

{ icmdidSetAsDef, "SetAsDef" },

{ icmdidCreateCmdFile, "CreateCmdFile" },

{ icmdCancel, "Cancel" },

{ icmdNewDatabase, "NewDatabase" },

{ icmdNewUser, "NewUser" },

{ icmdNewRole, "NewRole" },

{ icmdChangeLogin, "ChangeLogin" },

{ icmdNewView, "NewView" },

{ icmdModifyConnection, "ModifyConnection" },

{ icmdDisconnect, "Disconnect" },

{ icmdCopyScript, "CopyScript" },

{ icmdAddSCC, "AddSCC" },

{ icmdRemoveSCC, "RemoveSCC" },

{ icmdGetLatest, "GetLatest" },

{ icmdCheckOut, "CheckOut" },

{ icmdCheckIn, "CheckIn" },

{ icmdUndoCheckOut, "UndoCheckOut" },

{ icmdAddItemSCC, "AddItemSCC" },

{ icmdNewPackageSpec, "NewPackageSpec" },

{ icmdNewPackageBody, "NewPackageBody" },

{ icmdInsertSQL, "InsertSQL" },

{ icmdRunSelection, "RunSelection" },

{ icmdUpdateScript, "UpdateScript" },

{ icmdNewScript, "NewScript" },

{ icmdNewFunction, "NewFunction" },

{ icmdNewTableFunction, "NewTableFunction" },

{ icmdNewInlineFunction, "NewInlineFunction" },

{ icmdAddDiagram, "AddDiagram" },

{ icmdAddTable, "AddTable" },

{ icmdAddSynonym, "AddSynonym" },

{ icmdAddView, "AddView" },

{ icmdAddProcedure, "AddProcedure" },

{ icmdAddFunction, "AddFunction" },

{ icmdAddTableFunction, "AddTableFunction" },

{ icmdAddInlineFunction, "AddInlineFunction" },

{ icmdAddPkgSpec, "AddPkgSpec" },

{ icmdAddPkgBody, "AddPkgBody" },

{ icmdAddTrigger, "AddTrigger" },

{ icmdExportData, "ExportData" },

{ icmdDbnsVcsAdd, "DbnsVcsAdd" },

{ icmdDbnsVcsRemove, "DbnsVcsRemove" },

{ icmdDbnsVcsCheckout, "DbnsVcsCheckout" },

{ icmdDbnsVcsUndoCheckout, "DbnsVcsUndoCheckout" },

{ icmdDbnsVcsCheckin, "DbnsVcsCheckin" },

{ icmdSERetrieveData, "SERetrieveData" },

{ icmdSEEditTextObject, "SEEditTextObject" },

{ icmdDesignSQLBlock, "DesignSQLBlock" },

{ icmdRegisterSQLInstance, "RegisterSQLInstance" },

{ icmdUnregisterSQLInstance, "UnregisterSQLInstance" },

{ cmdidCommandWindowSaveScript, "CommandWindowSaveScript" },

{ cmdidCommandWindowRunScript, "CommandWindowRunScript" },

{ cmdidCommandWindowCursorUp, "CommandWindowCursorUp" },

{ cmdidCommandWindowCursorDown, "CommandWindowCursorDown" },

{ cmdidCommandWindowCursorLeft, "CommandWindowCursorLeft" },

{ cmdidCommandWindowCursorRight, "CommandWindowCursorRight" },

{ cmdidCommandWindowHistoryUp, "CommandWindowHistoryUp" },

{ cmdidCommandWindowHistoryDown, "CommandWindowHistoryDown" },

];

const enumName[] VSStd97CmdID\_names =

[

{ cmdidAlignBottom, "AlignBottom" },

{ cmdidAlignHorizontalCenters, "AlignHorizontalCenters" },

{ cmdidAlignLeft, "AlignLeft" },

{ cmdidAlignRight, "AlignRight" },

{ cmdidAlignToGrid, "AlignToGrid" },

{ cmdidAlignTop, "AlignTop" },

{ cmdidAlignVerticalCenters, "AlignVerticalCenters" },

{ cmdidArrangeBottom, "ArrangeBottom" },

{ cmdidArrangeRight, "ArrangeRight" },

{ cmdidBringForward, "BringForward" },

{ cmdidBringToFront, "BringToFront" },

{ cmdidCenterHorizontally, "CenterHorizontally" },

{ cmdidCenterVertically, "CenterVertically" },

{ cmdidCode, "Code" },

{ cmdidCopy, "Copy" },

{ cmdidCut, "Cut" },

{ cmdidDelete, "Delete" },

{ cmdidFontName, "FontName" },

{ cmdidFontSize, "FontSize" },

{ cmdidGroup, "Group" },

{ cmdidHorizSpaceConcatenate, "HorizSpaceConcatenate" },

{ cmdidHorizSpaceDecrease, "HorizSpaceDecrease" },

{ cmdidHorizSpaceIncrease, "HorizSpaceIncrease" },

{ cmdidHorizSpaceMakeEqual, "HorizSpaceMakeEqual" },

{ cmdidInsertObject, "InsertObject" },

{ cmdidPaste, "Paste" },

{ cmdidPrint, "Print" },

{ cmdidProperties, "Properties" },

{ cmdidRedo, "Redo" },

{ cmdidMultiLevelRedo, "MultiLevelRedo" },

{ cmdidSelectAll, "SelectAll" },

{ cmdidSendBackward, "SendBackward" },

{ cmdidSendToBack, "SendToBack" },

{ cmdidShowTable, "ShowTable" },

{ cmdidSizeToControl, "SizeToControl" },

{ cmdidSizeToControlHeight, "SizeToControlHeight" },

{ cmdidSizeToControlWidth, "SizeToControlWidth" },

{ cmdidSizeToFit, "SizeToFit" },

{ cmdidSizeToGrid, "SizeToGrid" },

{ cmdidSnapToGrid, "SnapToGrid" },

{ cmdidTabOrder, "TabOrder" },

{ cmdidToolbox, "Toolbox" },

{ cmdidUndo, "Undo" },

{ cmdidMultiLevelUndo, "MultiLevelUndo" },

{ cmdidUngroup, "Ungroup" },

{ cmdidVertSpaceConcatenate, "VertSpaceConcatenate" },

{ cmdidVertSpaceDecrease, "VertSpaceDecrease" },

{ cmdidVertSpaceIncrease, "VertSpaceIncrease" },

{ cmdidVertSpaceMakeEqual, "VertSpaceMakeEqual" },

{ cmdidZoomPercent, "ZoomPercent" },

{ cmdidBackColor, "BackColor" },

{ cmdidBold, "Bold" },

{ cmdidBorderColor, "BorderColor" },

{ cmdidBorderDashDot, "BorderDashDot" },

{ cmdidBorderDashDotDot, "BorderDashDotDot" },

{ cmdidBorderDashes, "BorderDashes" },

{ cmdidBorderDots, "BorderDots" },

{ cmdidBorderShortDashes, "BorderShortDashes" },

{ cmdidBorderSolid, "BorderSolid" },

{ cmdidBorderSparseDots, "BorderSparseDots" },

{ cmdidBorderWidth1, "BorderWidth1" },

{ cmdidBorderWidth2, "BorderWidth2" },

{ cmdidBorderWidth3, "BorderWidth3" },

{ cmdidBorderWidth4, "BorderWidth4" },

{ cmdidBorderWidth5, "BorderWidth5" },

{ cmdidBorderWidth6, "BorderWidth6" },

{ cmdidBorderWidthHairline, "BorderWidthHairline" },

{ cmdidFlat, "Flat" },

{ cmdidForeColor, "ForeColor" },

{ cmdidItalic, "Italic" },

{ cmdidJustifyCenter, "JustifyCenter" },

{ cmdidJustifyGeneral, "JustifyGeneral" },

{ cmdidJustifyLeft, "JustifyLeft" },

{ cmdidJustifyRight, "JustifyRight" },

{ cmdidRaised, "Raised" },

{ cmdidSunken, "Sunken" },

{ cmdidUnderline, "Underline" },

{ cmdidChiseled, "Chiseled" },

{ cmdidEtched, "Etched" },

{ cmdidShadowed, "Shadowed" },

{ cmdidCompDebug1, "CompDebug1" },

{ cmdidCompDebug2, "CompDebug2" },

{ cmdidCompDebug3, "CompDebug3" },

{ cmdidCompDebug4, "CompDebug4" },

{ cmdidCompDebug5, "CompDebug5" },

{ cmdidCompDebug6, "CompDebug6" },

{ cmdidCompDebug7, "CompDebug7" },

{ cmdidCompDebug8, "CompDebug8" },

{ cmdidCompDebug9, "CompDebug9" },

{ cmdidCompDebug10, "CompDebug10" },

{ cmdidCompDebug11, "CompDebug11" },

{ cmdidCompDebug12, "CompDebug12" },

{ cmdidCompDebug13, "CompDebug13" },

{ cmdidCompDebug14, "CompDebug14" },

{ cmdidCompDebug15, "CompDebug15" },

{ cmdidExistingSchemaEdit, "ExistingSchemaEdit" },

{ cmdidFind, "Find" },

{ cmdidGetZoom, "GetZoom" },

{ cmdidQueryOpenDesign, "QueryOpenDesign" },

{ cmdidQueryOpenNew, "QueryOpenNew" },

{ cmdidSingleTableDesign, "SingleTableDesign" },

{ cmdidSingleTableNew, "SingleTableNew" },

{ cmdidShowGrid, "ShowGrid" },

{ cmdidNewTable, "NewTable" },

{ cmdidCollapsedView, "CollapsedView" },

{ cmdidFieldView, "FieldView" },

{ cmdidVerifySQL, "VerifySQL" },

{ cmdidHideTable, "HideTable" },

{ cmdidPrimaryKey, "PrimaryKey" },

{ cmdidSave, "Save" },

{ cmdidSaveAs, "SaveAs" },

{ cmdidSortAscending, "SortAscending" },

{ cmdidSortDescending, "SortDescending" },

{ cmdidAppendQuery, "AppendQuery" },

{ cmdidCrosstabQuery, "CrosstabQuery" },

{ cmdidDeleteQuery, "DeleteQuery" },

{ cmdidMakeTableQuery, "MakeTableQuery" },

{ cmdidSelectQuery, "SelectQuery" },

{ cmdidUpdateQuery, "UpdateQuery" },

{ cmdidParameters, "Parameters" },

{ cmdidTotals, "Totals" },

{ cmdidViewCollapsed, "ViewCollapsed" },

{ cmdidViewFieldList, "ViewFieldList" },

{ cmdidViewKeys, "ViewKeys" },

{ cmdidViewGrid, "ViewGrid" },

{ cmdidInnerJoin, "InnerJoin" },

{ cmdidRightOuterJoin, "RightOuterJoin" },

{ cmdidLeftOuterJoin, "LeftOuterJoin" },

{ cmdidFullOuterJoin, "FullOuterJoin" },

{ cmdidUnionJoin, "UnionJoin" },

{ cmdidShowSQLPane, "ShowSQLPane" },

{ cmdidShowGraphicalPane, "ShowGraphicalPane" },

{ cmdidShowDataPane, "ShowDataPane" },

{ cmdidShowQBEPane, "ShowQBEPane" },

{ cmdidSelectAllFields, "SelectAllFields" },

{ cmdidOLEObjectMenuButton, "OLEObjectMenuButton" },

{ cmdidObjectVerbList0, "ObjectVerbList0" },

{ cmdidObjectVerbList1, "ObjectVerbList1" },

{ cmdidObjectVerbList2, "ObjectVerbList2" },

{ cmdidObjectVerbList3, "ObjectVerbList3" },

{ cmdidObjectVerbList4, "ObjectVerbList4" },

{ cmdidObjectVerbList5, "ObjectVerbList5" },

{ cmdidObjectVerbList6, "ObjectVerbList6" },

{ cmdidObjectVerbList7, "ObjectVerbList7" },

{ cmdidObjectVerbList8, "ObjectVerbList8" },

{ cmdidObjectVerbList9, "ObjectVerbList9" },

{ cmdidConvertObject, "ConvertObject" },

{ cmdidCustomControl, "CustomControl" },

{ cmdidCustomizeItem, "CustomizeItem" },

{ cmdidRename, "Rename" },

{ cmdidImport, "Import" },

{ cmdidNewPage, "NewPage" },

{ cmdidMove, "Move" },

{ cmdidCancel, "Cancel" },

{ cmdidFont, "Font" },

{ cmdidExpandLinks, "ExpandLinks" },

{ cmdidExpandImages, "ExpandImages" },

{ cmdidExpandPages, "ExpandPages" },

{ cmdidRefocusDiagram, "RefocusDiagram" },

{ cmdidTransitiveClosure, "TransitiveClosure" },

{ cmdidCenterDiagram, "CenterDiagram" },

{ cmdidZoomIn, "ZoomIn" },

{ cmdidZoomOut, "ZoomOut" },

{ cmdidRemoveFilter, "RemoveFilter" },

{ cmdidHidePane, "HidePane" },

{ cmdidDeleteTable, "DeleteTable" },

{ cmdidDeleteRelationship, "DeleteRelationship" },

{ cmdidRemove, "Remove" },

{ cmdidJoinLeftAll, "JoinLeftAll" },

{ cmdidJoinRightAll, "JoinRightAll" },

{ cmdidAddToOutput, "AddToOutput" },

{ cmdidOtherQuery, "OtherQuery" },

{ cmdidGenerateChangeScript, "GenerateChangeScript" },

{ cmdidSaveSelection, "SaveSelection" },

{ cmdidAutojoinCurrent, "AutojoinCurrent" },

{ cmdidAutojoinAlways, "AutojoinAlways" },

{ cmdidEditPage, "EditPage" },

{ cmdidViewLinks, "ViewLinks" },

{ cmdidStop, "Stop" },

{ cmdidPause, "Pause" },

{ cmdidResume, "Resume" },

{ cmdidFilterDiagram, "FilterDiagram" },

{ cmdidShowAllObjects, "ShowAllObjects" },

{ cmdidShowApplications, "ShowApplications" },

{ cmdidShowOtherObjects, "ShowOtherObjects" },

{ cmdidShowPrimRelationships, "ShowPrimRelationships" },

{ cmdidExpand, "Expand" },

{ cmdidCollapse, "Collapse" },

{ cmdidRefresh, "Refresh" },

{ cmdidLayout, "Layout" },

{ cmdidShowResources, "ShowResources" },

{ cmdidInsertHTMLWizard, "InsertHTMLWizard" },

{ cmdidShowDownloads, "ShowDownloads" },

{ cmdidShowExternals, "ShowExternals" },

{ cmdidShowInBoundLinks, "ShowInBoundLinks" },

{ cmdidShowOutBoundLinks, "ShowOutBoundLinks" },

{ cmdidShowInAndOutBoundLinks, "ShowInAndOutBoundLinks" },

{ cmdidPreview, "Preview" },

{ cmdidOpenWith, "OpenWith" },

{ cmdidShowPages, "ShowPages" },

{ cmdidRunQuery, "RunQuery" },

{ cmdidClearQuery, "ClearQuery" },

{ cmdidRecordFirst, "RecordFirst" },

{ cmdidRecordLast, "RecordLast" },

{ cmdidRecordNext, "RecordNext" },

{ cmdidRecordPrevious, "RecordPrevious" },

{ cmdidRecordGoto, "RecordGoto" },

{ cmdidRecordNew, "RecordNew" },

{ cmdidInsertNewMenu, "InsertNewMenu" },

{ cmdidInsertSeparator, "InsertSeparator" },

{ cmdidEditMenuNames, "EditMenuNames" },

{ cmdidDebugExplorer, "DebugExplorer" },

{ cmdidDebugProcesses, "DebugProcesses" },

{ cmdidViewThreadsWindow, "ViewThreadsWindow" },

{ cmdidWindowUIList, "WindowUIList" },

{ cmdidNewProject, "NewProject" },

{ cmdidOpenProject, "OpenProject" },

{ cmdidOpenSolution, "OpenSolution" },

{ cmdidCloseSolution, "CloseSolution" },

{ cmdidAddNewItem, "AddNewItem" },

{ cmdidFileNew, "FileNew" },

{ cmdidFileOpen, "FileOpen" },

{ cmdidFileClose, "FileClose" },

{ cmdidSaveSolution, "SaveSolution" },

{ cmdidSaveSolutionAs, "SaveSolutionAs" },

{ cmdidSaveProjectItemAs, "SaveProjectItemAs" },

{ cmdidPageSetup, "PageSetup" },

{ cmdidPrintPreview, "PrintPreview" },

{ cmdidExit, "Exit" },

{ cmdidReplace, "Replace" },

{ cmdidGoto, "Goto" },

{ cmdidPropertyPages, "PropertyPages" },

{ cmdidFullScreen, "FullScreen" },

{ cmdidProjectExplorer, "ProjectExplorer" },

{ cmdidPropertiesWindow, "PropertiesWindow" },

{ cmdidTaskListWindow, "TaskListWindow" },

{ cmdidOutputWindow, "OutputWindow" },

{ cmdidObjectBrowser, "ObjectBrowser" },

{ cmdidDocOutlineWindow, "DocOutlineWindow" },

{ cmdidImmediateWindow, "ImmediateWindow" },

{ cmdidWatchWindow, "WatchWindow" },

{ cmdidLocalsWindow, "LocalsWindow" },

{ cmdidCallStack, "CallStack" },

{ cmdidAddExistingItem, "AddExistingItem" },

{ cmdidNewFolder, "NewFolder" },

{ cmdidSetStartupProject, "SetStartupProject" },

{ cmdidProjectSettings, "ProjectSettings" },

{ cmdidStepInto, "StepInto" },

{ cmdidStepOver, "StepOver" },

{ cmdidStepOut, "StepOut" },

{ cmdidRunToCursor, "RunToCursor" },

{ cmdidAddWatch, "AddWatch" },

{ cmdidEditWatch, "EditWatch" },

{ cmdidQuickWatch, "QuickWatch" },

{ cmdidToggleBreakpoint, "ToggleBreakpoint" },

{ cmdidClearBreakpoints, "ClearBreakpoints" },

{ cmdidShowBreakpoints, "ShowBreakpoints" },

{ cmdidSetNextStatement, "SetNextStatement" },

{ cmdidShowNextStatement, "ShowNextStatement" },

{ cmdidEditBreakpoint, "EditBreakpoint" },

{ cmdidOpen, "Open" },

{ cmdidDetachDebugger, "DetachDebugger" },

{ cmdidCustomizeKeyboard, "CustomizeKeyboard" },

{ cmdidToolsOptions, "ToolsOptions" },

{ cmdidNewWindow, "NewWindow" },

{ cmdidSplit, "Split" },

{ cmdidCascade, "Cascade" },

{ cmdidTileHorz, "TileHorz" },

{ cmdidTileVert, "TileVert" },

{ cmdidTechSupport, "TechSupport" },

{ cmdidAbout, "About" },

{ cmdidDebugOptions, "DebugOptions" },

{ cmdidDeleteWatch, "DeleteWatch" },

{ cmdidCollapseWatch, "CollapseWatch" },

//        { cmdidFindSimplePattern, "FindSimplePattern" },

{ cmdidFindInFiles, "FindInFiles" },

{ cmdidReplaceInFiles, "ReplaceInFiles" },

{ cmdidNextLocation, "NextLocation" },

{ cmdidPreviousLocation, "PreviousLocation" },

{ cmdidGotoQuick, "GotoQuick" },

{ cmdidPbrsToggleStatus, "PbrsToggleStatus" },

{ cmdidPropbrsHide, "PropbrsHide" },

{ cmdidDockingView, "DockingView" },

{ cmdidHideActivePane, "HideActivePane" },

{ cmdidPaneNextTab, "PaneNextTab" },

{ cmdidPanePrevTab, "PanePrevTab" },

{ cmdidPaneCloseToolWindow, "PaneCloseToolWindow" },

{ cmdidPaneActivateDocWindow, "PaneActivateDocWindow" },

//        { cmdidDockingViewMDI, "DockingViewMDI" },

{ cmdidDockingViewFloater, "DockingViewFloater" },

{ cmdidAutoHideWindow, "AutoHideWindow" },

{ cmdidMoveToDropdownBar, "MoveToDropdownBar" },

{ cmdidFindCmd, "FindCmd" },

{ cmdidStart, "Start" },

{ cmdidRestart, "Restart" },

{ cmdidAddinManager, "AddinManager" },

{ cmdidMultiLevelUndoList, "MultiLevelUndoList" },

{ cmdidMultiLevelRedoList, "MultiLevelRedoList" },

{ cmdidToolboxAddTab, "ToolboxAddTab" },

{ cmdidToolboxDeleteTab, "ToolboxDeleteTab" },

{ cmdidToolboxRenameTab, "ToolboxRenameTab" },

{ cmdidToolboxTabMoveUp, "ToolboxTabMoveUp" },

{ cmdidToolboxTabMoveDown, "ToolboxTabMoveDown" },

{ cmdidToolboxRenameItem, "ToolboxRenameItem" },

{ cmdidToolboxListView, "ToolboxListView" },

{ cmdidSearchSetCombo, "SearchSetCombo" },

{ cmdidWindowUIGetList, "WindowUIGetList" },

{ cmdidInsertValuesQuery, "InsertValuesQuery" },

{ cmdidShowProperties, "ShowProperties" },

{ cmdidThreadSuspend, "ThreadSuspend" },

{ cmdidThreadResume, "ThreadResume" },

{ cmdidThreadSetFocus, "ThreadSetFocus" },

{ cmdidDisplayRadix, "DisplayRadix" },

{ cmdidOpenProjectItem, "OpenProjectItem" },

{ cmdidPaneNextPane, "PaneNextPane" },

{ cmdidPanePrevPane, "PanePrevPane" },

{ cmdidClearPane, "ClearPane" },

{ cmdidGotoErrorTag, "GotoErrorTag" },

//        { cmdidTaskListSortByCategory, "TaskListSortByCategory" },

//        { cmdidTaskListSortByFileLine, "TaskListSortByFileLine" },

//        { cmdidTaskListSortByPriority, "TaskListSortByPriority" },

//        { cmdidTaskListSortByDefaultSort, "TaskListSortByDefaultSort" },

//        { cmdidTaskListShowTooltip, "TaskListShowTooltip" },

//        { cmdidTaskListFilterByNothing, "TaskListFilterByNothing" },

{ cmdidCancelEZDrag, "CancelEZDrag" },

//        { cmdidTaskListFilterByCategoryCompiler, "TaskListFilterByCategoryCompiler" },

//        { cmdidTaskListFilterByCategoryComment, "TaskListFilterByCategoryComment" },

{ cmdidToolboxAddItem, "ToolboxAddItem" },

{ cmdidToolboxReset, "ToolboxReset" },

{ cmdidSaveProjectItem, "SaveProjectItem" },

{ cmdidViewForm, "ViewForm" },

{ cmdidViewCode, "ViewCode" },

{ cmdidPreviewInBrowser, "PreviewInBrowser" },

{ cmdidBrowseWith, "BrowseWith" },

{ cmdidSearchCombo, "SearchCombo" },

{ cmdidEditLabel, "EditLabel" },

{ cmdidExceptions, "Exceptions" },

//        { cmdidDefineViews, "DefineViews" },

{ cmdidToggleSelMode, "ToggleSelMode" },

{ cmdidToggleInsMode, "ToggleInsMode" },

{ cmdidLoadUnloadedProject, "LoadUnloadedProject" },

{ cmdidUnloadLoadedProject, "UnloadLoadedProject" },

{ cmdidElasticColumn, "ElasticColumn" },

{ cmdidHideColumn, "HideColumn" },

//        { cmdidTaskListPreviousView, "TaskListPreviousView" },

{ cmdidZoomDialog, "ZoomDialog" },

//        { cmdidFindHiddenText, "FindHiddenText" },

//        { cmdidFindMatchCase, "FindMatchCase" },

//        { cmdidFindWholeWord, "FindWholeWord" },

//        { cmdidFindRegularExpression, "FindRegularExpression" },

//        { cmdidFindBackwards, "FindBackwards" },

{ cmdidFindInSelection, "FindInSelection" },

{ cmdidFindStop, "FindStop" },

{ cmdidTaskListNextError, "TaskListNextError" },

{ cmdidTaskListPrevError, "TaskListPrevError" },

//        { cmdidTaskListFilterByCategoryUser, "TaskListFilterByCategoryUser" },

//        { cmdidTaskListFilterByCategoryShortcut, "TaskListFilterByCategoryShortcut" },

//        { cmdidTaskListFilterByCategoryHTML, "TaskListFilterByCategoryHTML" },

//        { cmdidTaskListFilterByCurrentFile, "TaskListFilterByCurrentFile" },

//        { cmdidTaskListFilterByChecked, "TaskListFilterByChecked" },

//        { cmdidTaskListFilterByUnchecked, "TaskListFilterByUnchecked" },

//        { cmdidTaskListSortByDescription, "TaskListSortByDescription" },

//        { cmdidTaskListSortByChecked, "TaskListSortByChecked" },

{ cmdidProjectReferences, "ProjectReferences" },

{ cmdidStartNoDebug, "StartNoDebug" },

{ cmdidLockControls, "LockControls" },

{ cmdidFindNext, "FindNext" },

{ cmdidFindPrev, "FindPrev" },

{ cmdidFindSelectedNext, "FindSelectedNext" },

{ cmdidFindSelectedPrev, "FindSelectedPrev" },

{ cmdidSearchGetList, "SearchGetList" },

{ cmdidInsertBreakpoint, "InsertBreakpoint" },

{ cmdidEnableBreakpoint, "EnableBreakpoint" },

{ cmdidF1Help, "F1Help" },

{ cmdidMoveToNextEZCntr, "MoveToNextEZCntr" },

{ cmdidNewProjectFromExisting, "NewProjectFromExisting" },

{ cmdidUpdateMarkerSpans, "UpdateMarkerSpans" },

{ cmdidMoveToPreviousEZCntr, "MoveToPreviousEZCntr" },

{ cmdidProjectProperties, "ProjectProperties" },

{ cmdidPropSheetOrProperties, "PropSheetOrProperties" },

{ cmdidTshellStep, "TshellStep" },

{ cmdidTshellRun, "TshellRun" },

{ cmdidMarkerCmd0, "MarkerCmd0" },

{ cmdidMarkerCmd1, "MarkerCmd1" },

{ cmdidMarkerCmd2, "MarkerCmd2" },

{ cmdidMarkerCmd3, "MarkerCmd3" },

{ cmdidMarkerCmd4, "MarkerCmd4" },

{ cmdidMarkerCmd5, "MarkerCmd5" },

{ cmdidMarkerCmd6, "MarkerCmd6" },

{ cmdidMarkerCmd7, "MarkerCmd7" },

{ cmdidMarkerCmd8, "MarkerCmd8" },

{ cmdidMarkerLast, "MarkerLast" },

{ cmdidMarkerCmd9, "MarkerCmd9" },

{ cmdidMarkerEnd, "MarkerEnd" },

{ cmdidReloadProject, "ReloadProject" },

{ cmdidUnloadProject, "UnloadProject" },

{ cmdidNewBlankSolution, "NewBlankSolution" },

{ cmdidSelectProjectTemplate, "SelectProjectTemplate" },

{ cmdidDetachAttachOutline, "DetachAttachOutline" },

{ cmdidShowHideOutline, "ShowHideOutline" },

{ cmdidSyncOutline, "SyncOutline" },

{ cmdidRunToCallstCursor, "RunToCallstCursor" },

{ cmdidNoCmdsAvailable, "NoCmdsAvailable" },

{ cmdidContextWindow, "ContextWindow" },

{ cmdidAlias, "Alias" },

{ cmdidGotoCommandLine, "GotoCommandLine" },

{ cmdidEvaluateExpression, "EvaluateExpression" },

{ cmdidImmediateMode, "ImmediateMode" },

{ cmdidEvaluateStatement, "EvaluateStatement" },

{ cmdidFindResultWindow1, "FindResultWindow1" },

{ cmdidFindResultWindow2, "FindResultWindow2" },

//        { cmdidOpenProjectFromWeb, "OpenProjectFromWeb" },

{ cmdidFileOpenFromWeb, "FileOpenFromWeb" },

{ cmdidFontNameGetList, "FontNameGetList" },

{ cmdidFontSizeGetList, "FontSizeGetList" },

{ cmdidRenameBookmark, "RenameBookmark" },

{ cmdidToggleBookmark, "ToggleBookmark" },

{ cmdidDeleteBookmark, "DeleteBookmark" },

{ cmdidBookmarkWindowGoToBookmark, "BookmarkWindowGoToBookmark" },

{ cmdidEnableBookmark, "EnableBookmark" },

{ cmdidNewBookmarkFolder, "NewBookmarkFolder" },

{ cmdidNextBookmarkFolder, "NextBookmarkFolder" },

{ cmdidPrevBookmarkFolder, "PrevBookmarkFolder" },

{ cmdidWindow1, "Window1" },

{ cmdidWindow2, "Window2" },

{ cmdidWindow3, "Window3" },

{ cmdidWindow4, "Window4" },

{ cmdidWindow5, "Window5" },

{ cmdidWindow6, "Window6" },

{ cmdidWindow7, "Window7" },

{ cmdidWindow8, "Window8" },

{ cmdidWindow9, "Window9" },

{ cmdidWindow10, "Window10" },

{ cmdidWindow11, "Window11" },

{ cmdidWindow12, "Window12" },

{ cmdidWindow13, "Window13" },

{ cmdidWindow14, "Window14" },

{ cmdidWindow15, "Window15" },

{ cmdidWindow16, "Window16" },

{ cmdidWindow17, "Window17" },

{ cmdidWindow18, "Window18" },

{ cmdidWindow19, "Window19" },

{ cmdidWindow20, "Window20" },

{ cmdidWindow21, "Window21" },

{ cmdidWindow22, "Window22" },

{ cmdidWindow23, "Window23" },

{ cmdidWindow24, "Window24" },

{ cmdidWindow25, "Window25" },

{ cmdidMoreWindows, "MoreWindows" },

{ cmdidAutoHideAllWindows, "AutoHideAllWindows" },

{ cmdidTaskListTaskHelp, "TaskListTaskHelp" },

{ cmdidClassView, "ClassView" },

{ cmdidMRUProj1, "MRUProj1" },

{ cmdidMRUProj2, "MRUProj2" },

{ cmdidMRUProj3, "MRUProj3" },

{ cmdidMRUProj4, "MRUProj4" },

{ cmdidMRUProj5, "MRUProj5" },

{ cmdidMRUProj6, "MRUProj6" },

{ cmdidMRUProj7, "MRUProj7" },

{ cmdidMRUProj8, "MRUProj8" },

{ cmdidMRUProj9, "MRUProj9" },

{ cmdidMRUProj10, "MRUProj10" },

{ cmdidMRUProj11, "MRUProj11" },

{ cmdidMRUProj12, "MRUProj12" },

{ cmdidMRUProj13, "MRUProj13" },

{ cmdidMRUProj14, "MRUProj14" },

{ cmdidMRUProj15, "MRUProj15" },

{ cmdidMRUProj16, "MRUProj16" },

{ cmdidMRUProj17, "MRUProj17" },

{ cmdidMRUProj18, "MRUProj18" },

{ cmdidMRUProj19, "MRUProj19" },

{ cmdidMRUProj20, "MRUProj20" },

{ cmdidMRUProj21, "MRUProj21" },

{ cmdidMRUProj22, "MRUProj22" },

{ cmdidMRUProj23, "MRUProj23" },

{ cmdidMRUProj24, "MRUProj24" },

{ cmdidMRUProj25, "MRUProj25" },

{ cmdidSplitNext, "SplitNext" },

{ cmdidSplitPrev, "SplitPrev" },

{ cmdidCloseAllDocuments, "CloseAllDocuments" },

{ cmdidNextDocument, "NextDocument" },

{ cmdidPrevDocument, "PrevDocument" },

{ cmdidTool1, "Tool1" },

{ cmdidTool2, "Tool2" },

{ cmdidTool3, "Tool3" },

{ cmdidTool4, "Tool4" },

{ cmdidTool5, "Tool5" },

{ cmdidTool6, "Tool6" },

{ cmdidTool7, "Tool7" },

{ cmdidTool8, "Tool8" },

{ cmdidTool9, "Tool9" },

{ cmdidTool10, "Tool10" },

{ cmdidTool11, "Tool11" },

{ cmdidTool12, "Tool12" },

{ cmdidTool13, "Tool13" },

{ cmdidTool14, "Tool14" },

{ cmdidTool15, "Tool15" },

{ cmdidTool16, "Tool16" },

{ cmdidTool17, "Tool17" },

{ cmdidTool18, "Tool18" },

{ cmdidTool19, "Tool19" },

{ cmdidTool20, "Tool20" },

{ cmdidTool21, "Tool21" },

{ cmdidTool22, "Tool22" },

{ cmdidTool23, "Tool23" },

{ cmdidTool24, "Tool24" },

{ cmdidExternalCommands, "ExternalCommands" },

{ cmdidPasteNextTBXCBItem, "PasteNextTBXCBItem" },

{ cmdidToolboxShowAllTabs, "ToolboxShowAllTabs" },

{ cmdidProjectDependencies, "ProjectDependencies" },

{ cmdidCloseDocument, "CloseDocument" },

{ cmdidToolboxSortItems, "ToolboxSortItems" },

{ cmdidViewBarView1, "ViewBarView1" },

{ cmdidViewBarView2, "ViewBarView2" },

{ cmdidViewBarView3, "ViewBarView3" },

{ cmdidViewBarView4, "ViewBarView4" },

{ cmdidViewBarView5, "ViewBarView5" },

{ cmdidViewBarView6, "ViewBarView6" },

{ cmdidViewBarView7, "ViewBarView7" },

{ cmdidViewBarView8, "ViewBarView8" },

{ cmdidViewBarView9, "ViewBarView9" },

{ cmdidViewBarView10, "ViewBarView10" },

{ cmdidViewBarView11, "ViewBarView11" },

{ cmdidViewBarView12, "ViewBarView12" },

{ cmdidViewBarView13, "ViewBarView13" },

{ cmdidViewBarView14, "ViewBarView14" },

{ cmdidViewBarView15, "ViewBarView15" },

{ cmdidManageIndexes, "ManageIndexes" },

{ cmdidViewBarView16, "ViewBarView16" },

{ cmdidManageRelationships, "ManageRelationships" },

{ cmdidViewBarView17, "ViewBarView17" },

{ cmdidManageConstraints, "ManageConstraints" },

{ cmdidViewBarView18, "ViewBarView18" },

{ cmdidViewBarView19, "ViewBarView19" },

//        { cmdidTaskListCustomView1, "TaskListCustomView1" },

//        { cmdidTaskListCustomView2, "TaskListCustomView2" },

{ cmdidViewBarView20, "ViewBarView20" },

//        { cmdidTaskListCustomView3, "TaskListCustomView3" },

{ cmdidViewBarView21, "ViewBarView21" },

//        { cmdidTaskListCustomView4, "TaskListCustomView4" },

{ cmdidViewBarView22, "ViewBarView22" },

{ cmdidViewBarView23, "ViewBarView23" },

//        { cmdidTaskListCustomView5, "TaskListCustomView5" },

//        { cmdidTaskListCustomView6, "TaskListCustomView6" },

{ cmdidViewBarView24, "ViewBarView24" },

//        { cmdidTaskListCustomView7, "TaskListCustomView7" },

{ cmdidSolutionCfg, "SolutionCfg" },

{ cmdidSolutionCfgGetList, "SolutionCfgGetList" },

//        { cmdidTaskListCustomView8, "TaskListCustomView8" },

//        { cmdidTaskListCustomView9, "TaskListCustomView9" },

//        { cmdidTaskListCustomView10, "TaskListCustomView10" },

//        { cmdidTaskListCustomView11, "TaskListCustomView11" },

//        { cmdidTaskListCustomView12, "TaskListCustomView12" },

//        { cmdidTaskListCustomView13, "TaskListCustomView13" },

//        { cmdidTaskListCustomView14, "TaskListCustomView14" },

//        { cmdidTaskListCustomView15, "TaskListCustomView15" },

//        { cmdidTaskListCustomView16, "TaskListCustomView16" },

//        { cmdidTaskListCustomView17, "TaskListCustomView17" },

//        { cmdidTaskListCustomView18, "TaskListCustomView18" },

//        { cmdidTaskListCustomView19, "TaskListCustomView19" },

//        { cmdidTaskListCustomView20, "TaskListCustomView20" },

//        { cmdidTaskListCustomView21, "TaskListCustomView21" },

//        { cmdidTaskListCustomView22, "TaskListCustomView22" },

//        { cmdidTaskListCustomView23, "TaskListCustomView23" },

//        { cmdidTaskListCustomView24, "TaskListCustomView24" },

//        { cmdidTaskListCustomView25, "TaskListCustomView25" },

//        { cmdidTaskListCustomView26, "TaskListCustomView26" },

//        { cmdidTaskListCustomView27, "TaskListCustomView27" },

//        { cmdidTaskListCustomView28, "TaskListCustomView28" },

//        { cmdidTaskListCustomView29, "TaskListCustomView29" },

//        { cmdidTaskListCustomView30, "TaskListCustomView30" },

//        { cmdidTaskListCustomView31, "TaskListCustomView31" },

//        { cmdidTaskListCustomView32, "TaskListCustomView32" },

//        { cmdidTaskListCustomView33, "TaskListCustomView33" },

//        { cmdidTaskListCustomView34, "TaskListCustomView34" },

//        { cmdidTaskListCustomView35, "TaskListCustomView35" },

//        { cmdidTaskListCustomView36, "TaskListCustomView36" },

//        { cmdidTaskListCustomView37, "TaskListCustomView37" },

//        { cmdidTaskListCustomView38, "TaskListCustomView38" },

//        { cmdidTaskListCustomView39, "TaskListCustomView39" },

//        { cmdidTaskListCustomView40, "TaskListCustomView40" },

//        { cmdidTaskListCustomView41, "TaskListCustomView41" },

//        { cmdidTaskListCustomView42, "TaskListCustomView42" },

//        { cmdidTaskListCustomView43, "TaskListCustomView43" },

//        { cmdidTaskListCustomView44, "TaskListCustomView44" },

//        { cmdidTaskListCustomView45, "TaskListCustomView45" },

//        { cmdidTaskListCustomView46, "TaskListCustomView46" },

//        { cmdidTaskListCustomView47, "TaskListCustomView47" },

//        { cmdidTaskListCustomView48, "TaskListCustomView48" },

//        { cmdidTaskListCustomView49, "TaskListCustomView49" },

//        { cmdidTaskListCustomView50, "TaskListCustomView50" },

{ cmdidWhiteSpace, "WhiteSpace" },

{ cmdidCommandWindow, "CommandWindow" },

{ cmdidCommandWindowMarkMode, "CommandWindowMarkMode" },

{ cmdidLogCommandWindow, "LogCommandWindow" },

{ cmdidShell, "Shell" },

{ cmdidSingleChar, "SingleChar" },

{ cmdidZeroOrMore, "ZeroOrMore" },

{ cmdidOneOrMore, "OneOrMore" },

{ cmdidBeginLine, "BeginLine" },

{ cmdidEndLine, "EndLine" },

{ cmdidBeginWord, "BeginWord" },

{ cmdidEndWord, "EndWord" },

{ cmdidCharInSet, "CharInSet" },

{ cmdidCharNotInSet, "CharNotInSet" },

{ cmdidOr, "Or" },

{ cmdidEscape, "Escape" },

{ cmdidTagExp, "TagExp" },

{ cmdidPatternMatchHelp, "PatternMatchHelp" },

{ cmdidRegExList, "RegExList" },

{ cmdidDebugReserved1, "DebugReserved1" },

{ cmdidAutosWindow, "AutosWindow" },

{ cmdidDebugReserved2, "DebugReserved2" },

{ cmdidThisWindow, "ThisWindow" },

{ cmdidDebugReserved3, "DebugReserved3" },

{ cmdidWildZeroOrMore, "WildZeroOrMore" },

{ cmdidWildSingleChar, "WildSingleChar" },

{ cmdidWildSingleDigit, "WildSingleDigit" },

{ cmdidWildCharInSet, "WildCharInSet" },

{ cmdidWildCharNotInSet, "WildCharNotInSet" },

{ cmdidFindWhatText, "FindWhatText" },

{ cmdidTaggedExp1, "TaggedExp1" },

{ cmdidTaggedExp2, "TaggedExp2" },

{ cmdidTaggedExp3, "TaggedExp3" },

{ cmdidTaggedExp4, "TaggedExp4" },

{ cmdidTaggedExp5, "TaggedExp5" },

{ cmdidTaggedExp6, "TaggedExp6" },

{ cmdidTaggedExp7, "TaggedExp7" },

{ cmdidTaggedExp8, "TaggedExp8" },

{ cmdidTaggedExp9, "TaggedExp9" },

{ cmdidEditorWidgetClick, "EditorWidgetClick" },

{ cmdidCmdWinUpdateAC, "CmdWinUpdateAC" },

{ cmdidSlnCfgMgr, "SlnCfgMgr" },

{ cmdidAddNewProject, "AddNewProject" },

{ cmdidAddExistingProject, "AddExistingProject" },

//        { cmdidAddExistingProjFromWeb, "AddExistingProjFromWeb" },

{ cmdidAutoHideContext1, "AutoHideContext1" },

{ cmdidAutoHideContext2, "AutoHideContext2" },

{ cmdidAutoHideContext3, "AutoHideContext3" },

{ cmdidAutoHideContext4, "AutoHideContext4" },

{ cmdidAutoHideContext5, "AutoHideContext5" },

{ cmdidAutoHideContext6, "AutoHideContext6" },

{ cmdidAutoHideContext7, "AutoHideContext7" },

{ cmdidAutoHideContext8, "AutoHideContext8" },

{ cmdidAutoHideContext9, "AutoHideContext9" },

{ cmdidAutoHideContext10, "AutoHideContext10" },

{ cmdidAutoHideContext11, "AutoHideContext11" },

{ cmdidAutoHideContext12, "AutoHideContext12" },

{ cmdidAutoHideContext13, "AutoHideContext13" },

{ cmdidAutoHideContext14, "AutoHideContext14" },

{ cmdidAutoHideContext15, "AutoHideContext15" },

{ cmdidAutoHideContext16, "AutoHideContext16" },

{ cmdidAutoHideContext17, "AutoHideContext17" },

{ cmdidAutoHideContext18, "AutoHideContext18" },

{ cmdidAutoHideContext19, "AutoHideContext19" },

{ cmdidAutoHideContext20, "AutoHideContext20" },

{ cmdidAutoHideContext21, "AutoHideContext21" },

{ cmdidAutoHideContext22, "AutoHideContext22" },

{ cmdidAutoHideContext23, "AutoHideContext23" },

{ cmdidAutoHideContext24, "AutoHideContext24" },

{ cmdidAutoHideContext25, "AutoHideContext25" },

{ cmdidAutoHideContext26, "AutoHideContext26" },

{ cmdidAutoHideContext27, "AutoHideContext27" },

{ cmdidAutoHideContext28, "AutoHideContext28" },

{ cmdidAutoHideContext29, "AutoHideContext29" },

{ cmdidAutoHideContext30, "AutoHideContext30" },

{ cmdidAutoHideContext31, "AutoHideContext31" },

{ cmdidAutoHideContext32, "AutoHideContext32" },

{ cmdidAutoHideContext33, "AutoHideContext33" },

{ cmdidShellNavBackward, "ShellNavBackward" },

{ cmdidShellNavForward, "ShellNavForward" },

{ cmdidShellNavigate1, "ShellNavigate1" },

{ cmdidShellNavigate2, "ShellNavigate2" },

{ cmdidShellNavigate3, "ShellNavigate3" },

{ cmdidShellNavigate4, "ShellNavigate4" },

{ cmdidShellNavigate5, "ShellNavigate5" },

{ cmdidShellNavigate6, "ShellNavigate6" },

{ cmdidShellNavigate7, "ShellNavigate7" },

{ cmdidShellNavigate8, "ShellNavigate8" },

{ cmdidShellNavigate9, "ShellNavigate9" },

{ cmdidShellNavigate10, "ShellNavigate10" },

{ cmdidShellNavigate11, "ShellNavigate11" },

{ cmdidShellNavigate12, "ShellNavigate12" },

{ cmdidShellNavigate13, "ShellNavigate13" },

{ cmdidShellNavigate14, "ShellNavigate14" },

{ cmdidShellNavigate15, "ShellNavigate15" },

{ cmdidShellNavigate16, "ShellNavigate16" },

{ cmdidShellNavigate17, "ShellNavigate17" },

{ cmdidShellNavigate18, "ShellNavigate18" },

{ cmdidShellNavigate19, "ShellNavigate19" },

{ cmdidShellNavigate20, "ShellNavigate20" },

{ cmdidShellNavigate21, "ShellNavigate21" },

{ cmdidShellNavigate22, "ShellNavigate22" },

{ cmdidShellNavigate23, "ShellNavigate23" },

{ cmdidShellNavigate24, "ShellNavigate24" },

{ cmdidShellNavigate25, "ShellNavigate25" },

{ cmdidShellNavigate26, "ShellNavigate26" },

{ cmdidShellNavigate27, "ShellNavigate27" },

{ cmdidShellNavigate28, "ShellNavigate28" },

{ cmdidShellNavigate29, "ShellNavigate29" },

{ cmdidShellNavigate30, "ShellNavigate30" },

{ cmdidShellNavigate31, "ShellNavigate31" },

{ cmdidShellNavigate32, "ShellNavigate32" },

{ cmdidShellNavigate33, "ShellNavigate33" },

{ cmdidShellWindowNavigate1, "ShellWindowNavigate1" },

{ cmdidShellWindowNavigate2, "ShellWindowNavigate2" },

{ cmdidShellWindowNavigate3, "ShellWindowNavigate3" },

{ cmdidShellWindowNavigate4, "ShellWindowNavigate4" },

{ cmdidShellWindowNavigate5, "ShellWindowNavigate5" },

{ cmdidShellWindowNavigate6, "ShellWindowNavigate6" },

{ cmdidShellWindowNavigate7, "ShellWindowNavigate7" },

{ cmdidShellWindowNavigate8, "ShellWindowNavigate8" },

{ cmdidShellWindowNavigate9, "ShellWindowNavigate9" },

{ cmdidShellWindowNavigate10, "ShellWindowNavigate10" },

{ cmdidShellWindowNavigate11, "ShellWindowNavigate11" },

{ cmdidShellWindowNavigate12, "ShellWindowNavigate12" },

{ cmdidShellWindowNavigate13, "ShellWindowNavigate13" },

{ cmdidShellWindowNavigate14, "ShellWindowNavigate14" },

{ cmdidShellWindowNavigate15, "ShellWindowNavigate15" },

{ cmdidShellWindowNavigate16, "ShellWindowNavigate16" },

{ cmdidShellWindowNavigate17, "ShellWindowNavigate17" },

{ cmdidShellWindowNavigate18, "ShellWindowNavigate18" },

{ cmdidShellWindowNavigate19, "ShellWindowNavigate19" },

{ cmdidShellWindowNavigate20, "ShellWindowNavigate20" },

{ cmdidShellWindowNavigate21, "ShellWindowNavigate21" },

{ cmdidShellWindowNavigate22, "ShellWindowNavigate22" },

{ cmdidShellWindowNavigate23, "ShellWindowNavigate23" },

{ cmdidShellWindowNavigate24, "ShellWindowNavigate24" },

{ cmdidShellWindowNavigate25, "ShellWindowNavigate25" },

{ cmdidShellWindowNavigate26, "ShellWindowNavigate26" },

{ cmdidShellWindowNavigate27, "ShellWindowNavigate27" },

{ cmdidShellWindowNavigate28, "ShellWindowNavigate28" },

{ cmdidShellWindowNavigate29, "ShellWindowNavigate29" },

{ cmdidShellWindowNavigate30, "ShellWindowNavigate30" },

{ cmdidShellWindowNavigate31, "ShellWindowNavigate31" },

{ cmdidShellWindowNavigate32, "ShellWindowNavigate32" },

{ cmdidShellWindowNavigate33, "ShellWindowNavigate33" },

{ cmdidOBSDoFind, "OBSDoFind" },

{ cmdidOBSMatchCase, "OBSMatchCase" },

{ cmdidOBSMatchSubString, "OBSMatchSubString" },

{ cmdidOBSMatchWholeWord, "OBSMatchWholeWord" },

{ cmdidOBSMatchPrefix, "OBSMatchPrefix" },

{ cmdidBuildSln, "BuildSln" },

{ cmdidRebuildSln, "RebuildSln" },

{ cmdidDeploySln, "DeploySln" },

{ cmdidCleanSln, "CleanSln" },

{ cmdidBuildSel, "BuildSel" },

{ cmdidRebuildSel, "RebuildSel" },

{ cmdidDeploySel, "DeploySel" },

{ cmdidCleanSel, "CleanSel" },

{ cmdidCancelBuild, "CancelBuild" },

{ cmdidBatchBuildDlg, "BatchBuildDlg" },

{ cmdidBuildCtx, "BuildCtx" },

{ cmdidRebuildCtx, "RebuildCtx" },

{ cmdidDeployCtx, "DeployCtx" },

{ cmdidCleanCtx, "CleanCtx" },

{ cmdidQryManageIndexes, "QryManageIndexes" },

{ cmdidPrintDefault, "PrintDefault" },

//        { cmdidBrowseDoc, "BrowseDoc" },

{ cmdidShowStartPage, "ShowStartPage" },

{ cmdidMRUFile1, "MRUFile1" },

{ cmdidMRUFile2, "MRUFile2" },

{ cmdidMRUFile3, "MRUFile3" },

{ cmdidMRUFile4, "MRUFile4" },

{ cmdidMRUFile5, "MRUFile5" },

{ cmdidMRUFile6, "MRUFile6" },

{ cmdidMRUFile7, "MRUFile7" },

{ cmdidMRUFile8, "MRUFile8" },

{ cmdidMRUFile9, "MRUFile9" },

{ cmdidMRUFile10, "MRUFile10" },

{ cmdidMRUFile11, "MRUFile11" },

{ cmdidMRUFile12, "MRUFile12" },

{ cmdidMRUFile13, "MRUFile13" },

{ cmdidMRUFile14, "MRUFile14" },

{ cmdidMRUFile15, "MRUFile15" },

{ cmdidMRUFile16, "MRUFile16" },

{ cmdidMRUFile17, "MRUFile17" },

{ cmdidMRUFile18, "MRUFile18" },

{ cmdidMRUFile19, "MRUFile19" },

{ cmdidMRUFile20, "MRUFile20" },

{ cmdidMRUFile21, "MRUFile21" },

{ cmdidMRUFile22, "MRUFile22" },

{ cmdidMRUFile23, "MRUFile23" },

{ cmdidMRUFile24, "MRUFile24" },

{ cmdidMRUFile25, "MRUFile25" },

{ cmdidExtToolsCurPath, "ExtToolsCurPath" },

{ cmdidExtToolsCurDir, "ExtToolsCurDir" },

{ cmdidExtToolsCurFileName, "ExtToolsCurFileName" },

{ cmdidExtToolsCurExtension, "ExtToolsCurExtension" },

{ cmdidExtToolsProjDir, "ExtToolsProjDir" },

{ cmdidExtToolsProjFileName, "ExtToolsProjFileName" },

{ cmdidExtToolsSlnDir, "ExtToolsSlnDir" },

{ cmdidExtToolsSlnFileName, "ExtToolsSlnFileName" },

{ cmdidGotoDefn, "GotoDefn" },

{ cmdidGotoDecl, "GotoDecl" },

{ cmdidBrowseDefn, "BrowseDefn" },

{ cmdidSyncClassView, "SyncClassView" },

{ cmdidShowMembers, "ShowMembers" },

{ cmdidShowBases, "ShowBases" },

{ cmdidShowDerived, "ShowDerived" },

{ cmdidShowDefns, "ShowDefns" },

{ cmdidShowRefs, "ShowRefs" },

{ cmdidShowCallers, "ShowCallers" },

{ cmdidShowCallees, "ShowCallees" },

{ cmdidAddClass, "AddClass" },

{ cmdidAddNestedClass, "AddNestedClass" },

{ cmdidAddInterface, "AddInterface" },

{ cmdidAddMethod, "AddMethod" },

{ cmdidAddProperty, "AddProperty" },

{ cmdidAddEvent, "AddEvent" },

{ cmdidAddVariable, "AddVariable" },

{ cmdidImplementInterface, "ImplementInterface" },

{ cmdidOverride, "Override" },

{ cmdidAddFunction, "AddFunction" },

{ cmdidAddConnectionPoint, "AddConnectionPoint" },

{ cmdidAddIndexer, "AddIndexer" },

{ cmdidBuildOrder, "BuildOrder" },

{ cmdidSaveOptions, "SaveOptions" },

{ cmdidOBShowHidden, "OBShowHidden" },

{ cmdidOBEnableGrouping, "OBEnableGrouping" },

{ cmdidOBSetGroupingCriteria, "OBSetGroupingCriteria" },

{ cmdidOBBack, "OBBack" },

{ cmdidOBForward, "OBForward" },

{ cmdidOBShowPackages, "OBShowPackages" },

{ cmdidOBSearchCombo, "OBSearchCombo" },

{ cmdidOBSearchOptWholeWord, "OBSearchOptWholeWord" },

{ cmdidOBSearchOptSubstring, "OBSearchOptSubstring" },

{ cmdidOBSearchOptPrefix, "OBSearchOptPrefix" },

{ cmdidOBSearchOptCaseSensitive, "OBSearchOptCaseSensitive" },

{ cmdidCVGroupingNone, "CVGroupingNone" },

{ cmdidCVGroupingSortOnly, "CVGroupingSortOnly" },

{ cmdidCVGroupingGrouped, "CVGroupingGrouped" },

{ cmdidCVShowPackages, "CVShowPackages" },

{ cmdidCVNewFolder, "CVNewFolder" },

{ cmdidCVGroupingSortAccess, "CVGroupingSortAccess" },

{ cmdidObjectSearch, "ObjectSearch" },

{ cmdidObjectSearchResults, "ObjectSearchResults" },

{ cmdidBuild1, "Build1" },

{ cmdidBuild2, "Build2" },

{ cmdidBuild3, "Build3" },

{ cmdidBuild4, "Build4" },

{ cmdidBuild5, "Build5" },

{ cmdidBuild6, "Build6" },

{ cmdidBuild7, "Build7" },

{ cmdidBuild8, "Build8" },

{ cmdidBuild9, "Build9" },

{ cmdidBuildLast, "BuildLast" },

{ cmdidRebuild1, "Rebuild1" },

{ cmdidRebuild2, "Rebuild2" },

{ cmdidRebuild3, "Rebuild3" },

{ cmdidRebuild4, "Rebuild4" },

{ cmdidRebuild5, "Rebuild5" },

{ cmdidRebuild6, "Rebuild6" },

{ cmdidRebuild7, "Rebuild7" },

{ cmdidRebuild8, "Rebuild8" },

{ cmdidRebuild9, "Rebuild9" },

{ cmdidRebuildLast, "RebuildLast" },

{ cmdidClean1, "Clean1" },

{ cmdidClean2, "Clean2" },

{ cmdidClean3, "Clean3" },

{ cmdidClean4, "Clean4" },

{ cmdidClean5, "Clean5" },

{ cmdidClean6, "Clean6" },

{ cmdidClean7, "Clean7" },

{ cmdidClean8, "Clean8" },

{ cmdidClean9, "Clean9" },

{ cmdidCleanLast, "CleanLast" },

{ cmdidDeploy1, "Deploy1" },

{ cmdidDeploy2, "Deploy2" },

{ cmdidDeploy3, "Deploy3" },

{ cmdidDeploy4, "Deploy4" },

{ cmdidDeploy5, "Deploy5" },

{ cmdidDeploy6, "Deploy6" },

{ cmdidDeploy7, "Deploy7" },

{ cmdidDeploy8, "Deploy8" },

{ cmdidDeploy9, "Deploy9" },

{ cmdidDeployLast, "DeployLast" },

{ cmdidBuildProjPicker, "BuildProjPicker" },

{ cmdidRebuildProjPicker, "RebuildProjPicker" },

{ cmdidCleanProjPicker, "CleanProjPicker" },

{ cmdidDeployProjPicker, "DeployProjPicker" },

{ cmdidResourceView, "ResourceView" },

//        { cmdidShowHomePage, "ShowHomePage" },

{ cmdidEditMenuIDs, "EditMenuIDs" },

{ cmdidLineBreak, "LineBreak" },

{ cmdidCPPIdentifier, "CPPIdentifier" },

{ cmdidQuotedString, "QuotedString" },

{ cmdidSpaceOrTab, "SpaceOrTab" },

{ cmdidInteger, "Integer" },

{ cmdidCustomizeToolbars, "CustomizeToolbars" },

{ cmdidMoveToTop, "MoveToTop" },

{ cmdidWindowHelp, "WindowHelp" },

{ cmdidViewPopup, "ViewPopup" },

{ cmdidCheckMnemonics, "CheckMnemonics" },

{ cmdidPRSortAlphabeticaly, "PRSortAlphabeticaly" },

{ cmdidPRSortByCategory, "PRSortByCategory" },

{ cmdidViewNextTab, "ViewNextTab" },

{ cmdidCheckForUpdates, "CheckForUpdates" },

{ cmdidBrowser1, "Browser1" },

{ cmdidBrowser2, "Browser2" },

{ cmdidBrowser3, "Browser3" },

{ cmdidBrowser4, "Browser4" },

{ cmdidBrowser5, "Browser5" },

{ cmdidBrowser6, "Browser6" },

{ cmdidBrowser7, "Browser7" },

{ cmdidBrowser8, "Browser8" },

{ cmdidBrowser9, "Browser9" },

{ cmdidBrowser10, "Browser10" },

{ cmdidBrowser11, "Browser11" },

{ cmdidOpenDropDownOpen, "OpenDropDownOpen" },

{ cmdidOpenDropDownOpenWith, "OpenDropDownOpenWith" },

{ cmdidToolsDebugProcesses, "ToolsDebugProcesses" },

{ cmdidPaneNextSubPane, "PaneNextSubPane" },

{ cmdidPanePrevSubPane, "PanePrevSubPane" },

{ cmdidMoveFileToProject1, "MoveFileToProject1" },

{ cmdidMoveFileToProject2, "MoveFileToProject2" },

{ cmdidMoveFileToProject3, "MoveFileToProject3" },

{ cmdidMoveFileToProject4, "MoveFileToProject4" },

{ cmdidMoveFileToProject5, "MoveFileToProject5" },

{ cmdidMoveFileToProject6, "MoveFileToProject6" },

{ cmdidMoveFileToProject7, "MoveFileToProject7" },

{ cmdidMoveFileToProject8, "MoveFileToProject8" },

{ cmdidMoveFileToProject9, "MoveFileToProject9" },

{ cmdidMoveFileToProjectLast, "MoveFileToProjectLast" },

{ cmdidMoveFileToProjectPick, "MoveFileToProjectPick" },

{ cmdidDefineSubset, "DefineSubset" },

{ cmdidSubsetCombo, "SubsetCombo" },

{ cmdidSubsetGetList, "SubsetGetList" },

{ cmdidOBSortObjectsAlpha, "OBSortObjectsAlpha" },

{ cmdidOBSortObjectsType, "OBSortObjectsType" },

{ cmdidOBSortObjectsAccess, "OBSortObjectsAccess" },

{ cmdidOBGroupObjectsType, "OBGroupObjectsType" },

{ cmdidOBGroupObjectsAccess, "OBGroupObjectsAccess" },

{ cmdidOBSortMembersAlpha, "OBSortMembersAlpha" },

{ cmdidOBSortMembersType, "OBSortMembersType" },

{ cmdidOBSortMembersAccess, "OBSortMembersAccess" },

{ cmdidPopBrowseContext, "PopBrowseContext" },

{ cmdidGotoRef, "GotoRef" },

{ cmdidOBSLookInReferences, "OBSLookInReferences" },

{ cmdidExtToolsTargetPath, "ExtToolsTargetPath" },

{ cmdidExtToolsTargetDir, "ExtToolsTargetDir" },

{ cmdidExtToolsTargetFileName, "ExtToolsTargetFileName" },

{ cmdidExtToolsTargetExtension, "ExtToolsTargetExtension" },

{ cmdidExtToolsCurLine, "ExtToolsCurLine" },

{ cmdidExtToolsCurCol, "ExtToolsCurCol" },

{ cmdidExtToolsCurText, "ExtToolsCurText" },

{ cmdidBrowseNext, "BrowseNext" },

{ cmdidBrowsePrev, "BrowsePrev" },

{ cmdidBrowseUnload, "BrowseUnload" },

{ cmdidQuickObjectSearch, "QuickObjectSearch" },

{ cmdidExpandAll, "ExpandAll" },

{ cmdidExtToolsBinDir, "ExtToolsBinDir" },

{ cmdidBookmarkWindow, "BookmarkWindow" },

{ cmdidCodeExpansionWindow, "CodeExpansionWindow" },

{ cmdidNextDocumentNav, "NextDocumentNav" },

{ cmdidPrevDocumentNav, "PrevDocumentNav" },

{ cmdidForwardBrowseContext, "ForwardBrowseContext" },

{ cmdidStandardMax, "StandardMax" },

{ cmdidFindReferences, "FindReferences" },

{ cmdidFormsFirst, "FormsFirst" },

{ cmdidFormsLast, "FormsLast" },

{ cmdidVBEFirst, "VBEFirst" },

{ cmdidZoom200, "Zoom200" },

{ cmdidZoom150, "Zoom150" },

{ cmdidZoom100, "Zoom100" },

{ cmdidZoom75, "Zoom75" },

{ cmdidZoom50, "Zoom50" },

{ cmdidZoom25, "Zoom25" },

{ cmdidZoom10, "Zoom10" },

{ cmdidVBELast, "VBELast" },

{ cmdidSterlingFirst, "SterlingFirst" },

{ cmdidSterlingLast, "SterlingLast" },

{ uieventidFirst, "uieventidFirst" },

{ uieventidSelectRegion, "uieventidSelectRegion" },

{ uieventidDrop, "uieventidDrop" },

{ uieventidLast, "uieventidLast" },

];

const enumName[] VsUIHierarchyWindowCmdIds\_names =

[

{ UIHWCMDID\_RightClick, "UIHWCMDID\_RightClick" },

{ UIHWCMDID\_DoubleClick, "UIHWCMDID\_DoubleClick" },

{ UIHWCMDID\_EnterKey, "UIHWCMDID\_EnterKey" },

{ UIHWCMDID\_StartLabelEdit, "UIHWCMDID\_StartLabelEdit" },

{ UIHWCMDID\_CommitLabelEdit, "UIHWCMDID\_CommitLabelEdit" },

{ UIHWCMDID\_CancelLabelEdit, "UIHWCMDID\_CancelLabelEdit" },

];

// removed in VS12 SDK

static if(!is(typeof(cmdidLoadSymbolsDisabled))) enum cmdidLoadSymbolsDisabled = 0x15c;

const enumName[] VSDebugCommandCmdIds\_names =

[

{ cmdidBreakpointsWindowShow, "cmdidBreakpointsWindowShow" },

{ cmdidDisasmWindowShow, "cmdidDisasmWindowShow" },

{ cmdidProgramToDebugShow, "cmdidProgramToDebugShow" },

{ cmdidRegisterWindowShow, "cmdidRegisterWindowShow" },

{ cmdidModulesWindowShow, "cmdidModulesWindowShow" },

{ cmdidApplyCodeChanges, "cmdidApplyCodeChanges" },

{ cmdidStopApplyCodeChanges, "cmdidStopApplyCodeChanges" },

{ cmdidGoToDisassembly, "cmdidGoToDisassembly" },

{ cmdidShowDebugOutput, "cmdidShowDebugOutput" },

{ cmdidStepUnitLine, "cmdidStepUnitLine" },

{ cmdidStepUnitStatement, "cmdidStepUnitStatement" },

{ cmdidStepUnitInstruction, "cmdidStepUnitInstruction" },

{ cmdidStepUnitList, "cmdidStepUnitList" },

{ cmdidStepUnitListEnum, "cmdidStepUnitListEnum" },

{ cmdidWriteCrashDump, "cmdidWriteCrashDump" },

{ cmdidProcessList, "cmdidProcessList" },

{ cmdidProcessListEnum, "cmdidProcessListEnum" },

{ cmdidThreadList, "cmdidThreadList" },

{ cmdidThreadListEnum, "cmdidThreadListEnum" },

{ cmdidStackFrameList, "cmdidStackFrameList" },

{ cmdidStackFrameListEnum, "cmdidStackFrameListEnum" },

{ cmdidDisableAllBreakpoints, "cmdidDisableAllBreakpoints" },

{ cmdidEnableAllBreakpoints, "cmdidEnableAllBreakpoints" },

{ cmdidToggleAllBreakpoints, "cmdidToggleAllBreakpoints" },

{ cmdidTerminateAll, "cmdidTerminateAll" },

{ cmdidSymbolOptions, "cmdidSymbolOptions" },

{ cmdidLoadSymbolsFromCurrentPath, "cmdidLoadSymbolsFromCurrentPath" },

{ cmdidSymbolLoadInfo, "cmdidSymbolLoadInfo" },

{ cmdidStopEvaluatingExpression, "cmdidStopEvaluatingExpression" },

{ cmdidAttachedProcsWindowShow, "cmdidAttachedProcsWindowShow" },

{ cmdidToggleFlaggedThreads, "cmdidToggleFlaggedThreads" },

{ cmdidThreadFlag, "cmdidThreadFlag" },

{ cmdidThreadUnflag, "cmdidThreadUnflag" },

{ cmdidJustMyCode, "cmdidJustMyCode" },

{ cmdidNewFileBreakpoint, "cmdidNewFileBreakpoint" },

{ cmdidNewFunctionBreakpoint, "cmdidNewFunctionBreakpoint" },

{ cmdidNewAddressBreakpoint, "cmdidNewAddressBreakpoint" },

{ cmdidNewDataBreakpoint, "cmdidNewDataBreakpoint" },

{ cmdidThreadUnflagAll, "cmdidThreadUnflagAll" },

{ cmdidInsertTracepoint, "cmdidInsertTracepoint" },

{ cmdidBreakpointLocation, "cmdidBreakpointLocation" },

{ cmdidBreakpointCondition, "cmdidBreakpointCondition" },

{ cmdidBreakpointHitCount, "cmdidBreakpointHitCount" },

{ cmdidBreakpointConstraints, "cmdidBreakpointConstraints" },

{ cmdidBreakpointAction, "cmdidBreakpointAction" },

{ cmdidCreateObjectID, "cmdidCreateObjectID" },

// not in VS11 SDK: { cmdidRunMacrosForBreakpointsJustHit, "cmdidRunMacrosForBreakpointsJustHit" },

{ cmdidCopyExpression, "cmdidCopyExpression" },

{ cmdidCopyValue, "cmdidCopyValue" },

{ cmdidDestroyObjectID, "cmdidDestroyObjectID" },

{ cmdidOutputOnException, "cmdidOutputOnException" },

{ cmdidOutputOnModuleLoad, "cmdidOutputOnModuleLoad" },

{ cmdidOutputOnModuleUnload, "cmdidOutputOnModuleUnload" },

{ cmdidOutputOnProcessDestroy, "cmdidOutputOnProcessDestroy" },

{ cmdidOutputOnThreadDestroy, "cmdidOutputOnThreadDestroy" },

{ cmdidOutputOnOutputDebugString, "cmdidOutputOnOutputDebugString" },

{ cmdidSingleProcStepInto, "cmdidSingleProcStepInto" },

{ cmdidSingleProcStepOver, "cmdidSingleProcStepOver" },

{ cmdidSingleProcStepOut, "cmdidSingleProcStepOut" },

{ cmdidToggleCurrentThreadFlag, "cmdidToggleCurrentThreadFlag" },

{ cmdidShowThreadIpIndicators, "cmdidShowThreadIpIndicators" },

{ cmdidLoadSymbolsFromPublic, "cmdidLoadSymbolsFromPublic" },

{ cmdidLoadSymbolsDisabled, "cmdidLoadSymbolsDisabled" },

{ cmdidOutputOnStepFilter, "cmdidOutputOnStepFilter" },

{ cmdidStepFilterToggle, "cmdidStepFilterToggle" },

{ cmdidShowStepIntoSpecificMenu, "cmdidShowStepIntoSpecificMenu" },

// See above for explanation of these constants...

{ cmdidMemoryWindowShow, "cmdidMemoryWindowShow" },

{ cmdidMemoryWindowShow1, "cmdidMemoryWindowShow1" },

{ cmdidMemoryWindowShow2, "cmdidMemoryWindowShow2" },

{ cmdidMemoryWindowShow3, "cmdidMemoryWindowShow3" },

{ cmdidMemoryWindowShow4, "cmdidMemoryWindowShow4" },

{ cmdidWatchWindowShow, "cmdidWatchWindowShow" },

{ cmdidWatchWindowShow1, "cmdidWatchWindowShow1" },

{ cmdidWatchWindowShow2, "cmdidWatchWindowShow2" },

{ cmdidWatchWindowShow3, "cmdidWatchWindowShow3" },

{ cmdidWatchWindowShow4, "cmdidWatchWindowShow4" },

// Breakpoint Window commands

{ cmdidBreakpointsWindowFirst, "cmdidBreakpointsWindowFirst" },

{ cmdidBreakpointsWindowLast, "cmdidBreakpointsWindowLast" },

{ cmdidBreakpointsWindowNewBreakpoint, "cmdidBreakpointsWindowNewBreakpoint" },

{ cmdidBreakpointsWindowNewGroup, "cmdidBreakpointsWindowNewGroup" },

{ cmdidBreakpointsWindowDelete, "cmdidBreakpointsWindowDelete" },

{ cmdidBreakpointsWindowProperties, "cmdidBreakpointsWindowProperties" },

{ cmdidBreakpointsWindowDefaultGroup, "cmdidBreakpointsWindowDefaultGroup" },

{ cmdidBreakpointsWindowGoToSource, "cmdidBreakpointsWindowGoToSource" },

{ cmdidBreakpointsWindowGoToDisassembly, "cmdidBreakpointsWindowGoToDisassembly" },

{ cmdidBreakpointsWindowGoToBreakpoint, "cmdidBreakpointsWindowGoToBreakpoint" },

{ cmdidBreakpointsWindowColumnName, "cmdidBreakpointsWindowColumnName" },

{ cmdidBreakpointsWindowColumnCondition, "cmdidBreakpointsWindowColumnCondition" },

{ cmdidBreakpointsWindowColumnHitCount, "cmdidBreakpointsWindowColumnHitCount" },

{ cmdidBreakpointsWindowColumnLanguage, "cmdidBreakpointsWindowColumnLanguage" },

{ cmdidBreakpointsWindowColumnFunction, "cmdidBreakpointsWindowColumnFunction" },

{ cmdidBreakpointsWindowColumnFile, "cmdidBreakpointsWindowColumnFile" },

{ cmdidBreakpointsWindowColumnAddress, "cmdidBreakpointsWindowColumnAddress" },

{ cmdidBreakpointsWindowColumnData, "cmdidBreakpointsWindowColumnData" },

{ cmdidBreakpointsWindowColumnProcess, "cmdidBreakpointsWindowColumnProcess" },

{ cmdidBreakpointsWindowColumnConstraints, "cmdidBreakpointsWindowColumnConstraints" },

{ cmdidBreakpointsWindowColumnAction, "cmdidBreakpointsWindowColumnAction" },

// Disassembly Window commands

{ cmdidGoToSource, "cmdidGoToSource" },

{ cmdidShowDisasmAddress, "cmdidShowDisasmAddress" },

{ cmdidShowDisasmSource, "cmdidShowDisasmSource" },

{ cmdidShowDisasmCodeBytes, "cmdidShowDisasmCodeBytes" },

{ cmdidShowDisasmSymbolNames, "cmdidShowDisasmSymbolNames" },

{ cmdidShowDisasmLineNumbers, "cmdidShowDisasmLineNumbers" },

{ cmdidShowDisasmToolbar, "cmdidShowDisasmToolbar" },

{ cmdidDisasmExpression, "cmdidDisasmExpression" },

{ cmdidToggleDisassembly, "cmdidToggleDisassembly" },

// Memory Window commands

{ cmdidMemoryExpression, "cmdidMemoryExpression" },

{ cmdidMemoryExpression1, "cmdidMemoryExpression1" },

{ cmdidMemoryExpression2, "cmdidMemoryExpression2" },

{ cmdidMemoryExpression3, "cmdidMemoryExpression3" },

{ cmdidMemoryExpression4, "cmdidMemoryExpression4" },

{ cmdidAutoReevaluate, "cmdidAutoReevaluate" },

{ cmdidAutoReevaluate1, "cmdidAutoReevaluate1" },

{ cmdidAutoReevaluate2, "cmdidAutoReevaluate2" },

{ cmdidAutoReevaluate3, "cmdidAutoReevaluate3" },

{ cmdidAutoReevaluate4, "cmdidAutoReevaluate4" },

{ cmdidMemoryColumns, "cmdidMemoryColumns" },

{ cmdidMemoryColumns1, "cmdidMemoryColumns1" },

{ cmdidMemoryColumns2, "cmdidMemoryColumns2" },

{ cmdidMemoryColumns3, "cmdidMemoryColumns3" },

{ cmdidMemoryColumns4, "cmdidMemoryColumns4" },

{ cmdidColCountList, "cmdidColCountList" },

{ cmdidColCountList1, "cmdidColCountList1" },

{ cmdidColCountList2, "cmdidColCountList2" },

{ cmdidColCountList3, "cmdidColCountList3" },

{ cmdidColCountList4, "cmdidColCountList4" },

// The following apply to all instances of the memory windows. If any of these

// are added to the toolbar, they must be made per-instance!

{ cmdidShowNoData, "cmdidShowNoData" },

{ cmdidOneByteInt, "cmdidOneByteInt" },

{ cmdidTwoByteInt, "cmdidTwoByteInt" },

{ cmdidFourByteInt, "cmdidFourByteInt" },

{ cmdidEightByteInt, "cmdidEightByteInt" },

{ cmdidFloat, "cmdidFloat" },

{ cmdidDouble, "cmdidDouble" },

{ cmdidFormatHex, "cmdidFormatHex" },

{ cmdidFormatSigned, "cmdidFormatSigned" },

{ cmdidFormatUnsigned, "cmdidFormatUnsigned" },

{ cmdidFormatBigEndian, "cmdidFormatBigEndian" },

{ cmdidShowNoText, "cmdidShowNoText" },

{ cmdidShowAnsiText, "cmdidShowAnsiText" },

{ cmdidShowUnicodeText, "cmdidShowUnicodeText" },

{ cmdidEditValue, "cmdidEditValue" },

{ cmdidShowToolbar, "cmdidShowToolbar" },

// MemoryView-specific commands. These are used internally by the MemoryView implementation.

{ cmdidStopInPlaceEdit, "cmdidStopInPlaceEdit" },

// Registers Window commands

{ cmdidRegisterWindowFirst, "cmdidRegisterWindowFirst" },

{ cmdidRegWinGroupFirst, "cmdidRegWinGroupFirst" },

{ cmdidRegWinGroupLast, "cmdidRegWinGroupLast" },

{ cmdidRegisterWindowLast, "cmdidRegisterWindowLast" },

// QuickWatch commands

{ cmdidQuickWatchFirst, "cmdidQuickWatchFirst" },

{ cmdidQuickWatchLast, "cmdidQuickWatchLast" },

// Modules Window commands

{ cmdidModulesWindowFirst, "cmdidModulesWindowFirst" },

{ cmdidModulesWindowLast, "cmdidModulesWindowLast" },

{ cmdidReloadSymbols, "cmdidReloadSymbols" },

{ cmdidShowAllModules, "cmdidShowAllModules" },

{ cmdidToggleUserCode, "cmdidToggleUserCode" },

// step into specific

{ cmdidStepIntoSpecificFirst, "cmdidStepIntoSpecificFirst" },

{ cmdidStepIntoSpecificLast, "cmdidStepIntoSpecificLast" },

// Call Stack commands

{ cmdidCallStackWindowFirst, "cmdidCallStackWindowFirst" },

{ cmdidCallStackWindowLast, "cmdidCallStackWindowLast" },

{ cmdidSetCurrentFrame, "cmdidSetCurrentFrame" },

{ cmdidCallStackValues, "cmdidCallStackValues" },

{ cmdidCallStackTypes, "cmdidCallStackTypes" },

{ cmdidCallStackNames, "cmdidCallStackNames" },

{ cmdidCallStackModules, "cmdidCallStackModules" },

{ cmdidCallStackLineOffset, "cmdidCallStackLineOffset" },

{ cmdidCallStackByteOffset, "cmdidCallStackByteOffset" },

{ cmdidCrossThreadCallStack, "cmdidCrossThreadCallStack" },

{ cmdidShowExternalCode, "cmdidShowExternalCode" },

{ cmdidUnwindFromException, "cmdidUnwindFromException" },

// Datatip commands

{ cmdidDatatipFirst, "cmdidDatatipFirst" },

{ cmdidDatatipLast, "cmdidDatatipLast" },

{ cmdidDatatipNoTransparency, "cmdidDatatipNoTransparency" },

{ cmdidDatatipLowTransparency, "cmdidDatatipLowTransparency" },

{ cmdidDatatipMedTransparency, "cmdidDatatipMedTransparency" },

{ cmdidDatatipHighTransparency, "cmdidDatatipHighTransparency" },

// Attached Processes Window commands

{ cmdidAttachedProcsWindowFirst, "cmdidAttachedProcsWindowFirst" },

{ cmdidAttachedProcsWindowLast, "cmdidAttachedProcsWindowLast" },

{ cmdidAttachedProcsStartProcess, "cmdidAttachedProcsStartProcess" },

{ cmdidAttachedProcsPauseProcess, "cmdidAttachedProcsPauseProcess" },

{ cmdidAttachedProcsStepIntoProcess, "cmdidAttachedProcsStepIntoProcess" },

{ cmdidAttachedProcsStepOverProcess, "cmdidAttachedProcsStepOverProcess" },

{ cmdidAttachedProcsStepOutProcess, "cmdidAttachedProcsStepOutProcess" },

{ cmdidAttachedProcsDetachProcess, "cmdidAttachedProcsDetachProcess" },

{ cmdidAttachedProcsTerminateProcess, "cmdidAttachedProcsTerminateProcess" },

{ cmdidAttachedProcsDetachOnStop, "cmdidAttachedProcsDetachOnStop" },

{ cmdidAttachedProcsColumnName, "cmdidAttachedProcsColumnName" },

{ cmdidAttachedProcsColumnID, "cmdidAttachedProcsColumnID" },

{ cmdidAttachedProcsColumnPath, "cmdidAttachedProcsColumnPath" },

{ cmdidAttachedProcsColumnTitle, "cmdidAttachedProcsColumnTitle" },

{ cmdidAttachedProcsColumnMachine, "cmdidAttachedProcsColumnMachine" },

{ cmdidAttachedProcsColumnState, "cmdidAttachedProcsColumnState" },

{ cmdidAttachedProcsColumnTransport, "cmdidAttachedProcsColumnTransport" },

{ cmdidAttachedProcsColumnTransportQualifier, "cmdidAttachedProcsColumnTransportQualifier" },

{ cmdidThreadIpMarkerSwitchContext, "cmdidThreadIpMarkerSwitchContext" },

{ cmdidThreadIpMarkerFlagUnflag, "cmdidThreadIpMarkerFlagUnflag" },

{ cmdidThreadIpMarkersSwitchContext, "cmdidThreadIpMarkersSwitchContext" },

{ cmdidThreadIpMarkersSwitchContextFirst, "cmdidThreadIpMarkersSwitchContextFirst" },

{ cmdidThreadIpMarkersSwitchContextLast, "cmdidThreadIpMarkersSwitchContextLast" },

{ cmdidThreadIpMarkersFlag, "cmdidThreadIpMarkersFlag" },

{ cmdidThreadIpMarkersFlagFirst, "cmdidThreadIpMarkersFlagFirst" },

{ cmdidThreadIpMarkersFlagLast, "cmdidThreadIpMarkersFlagLast" },

{ cmdidThreadIpMarkersUnflag, "cmdidThreadIpMarkersUnflag" },

{ cmdidThreadIpMarkersUnflagFirst, "cmdidThreadIpMarkersUnflagFirst" },

{ cmdidThreadIpMarkersUnflagLast, "cmdidThreadIpMarkersUnflagLast" },

// Command Window commands

// while all commands are available in the command window,

// these are not on any menus by default

//

{ cmdidCommandWindowFirst, "cmdidCommandWindowFirst" },

{ cmdidCommandWindowLast, "cmdidCommandWindowLast" },

{ cmdidListMemory, "cmdidListMemory" },

{ cmdidListCallStack, "cmdidListCallStack" },

{ cmdidListDisassembly, "cmdidListDisassembly" },

{ cmdidListRegisters, "cmdidListRegisters" },

{ cmdidListThreads, "cmdidListThreads" },

{ cmdidSetRadix, "cmdidSetRadix" },

{ cmdidSetCurrentThread, "cmdidSetCurrentThread" },

{ cmdidSetCurrentStackFrame, "cmdidSetCurrentStackFrame" },

{ cmdidListSource, "cmdidListSource" },

{ cmdidSymbolPath, "cmdidSymbolPath" },

{ cmdidListModules, "cmdidListModules" },

{ cmdidListProcesses, "cmdidListProcesses" },

{ cmdidSetCurrentProcess, "cmdidSetCurrentProcess" },

];

string enum\_string(in enumName[] names, uint cmd)

{

foreach(ref const(enumName) en; names)

if(en.id == cmd)

return en.name;

return "";

}

} // debug

string cmd2string(ref const(GUID) guidCmdGroup, uint cmdID)

{

string name;

debug

{

if(guidCmdGroup == CMDSETID\_StandardCommandSet2K)

{

name = enum\_string(VSStd2KCmdID\_names, cmdID);

}

else if(guidCmdGroup == CMDSETID\_StandardCommandSet97)

{

name = enum\_string(VSStd97CmdID\_names, cmdID);

}

else if(guidCmdGroup == GUID\_VsUIHierarchyWindowCmds)

{

name = enum\_string(VsUIHierarchyWindowCmdIds\_names, cmdID);

}

else if(guidCmdGroup == guidVSDebugCommand)

{

name = enum\_string(VSDebugCommandCmdIds\_names, cmdID);

}

}

if(name.length == 0)

name = GUID2utf8(guidCmdGroup) ~ ":" ~ format("%d", cmdID);

return name;

}

// This file is part of Visual D

//

// Visual D integrates the D programming language into Visual Studio

// Copyright (c) 2010 by Rainer Schuetze, All Rights Reserved

//

// Distributed under the Boost Software License, Version 1.0.

// See accompanying file LICENSE\_1\_0.txt or copy at <http://www.boost.org/LICENSE_1_0.txt>

module visuald.winctrl;

import visuald.windows;

import visuald.logutil;

import std.utf;

import std.string;

import std.array;

import std.exception;

import std.algorithm;

import sdk.port.base;

import sdk.win32.prsht;

import sdk.win32.commctrl;

private Widget[Widget] createdWindows; // collection of all windows with HWND to avoid garbage collection

alias AssociativeArray!(Widget, Widget) \_wa1; // fully instantiate type info

private HINSTANCE hInst;

private HFONT winFont;

LOGFONTW dialogLogFont = { lfHeight : -9, lfCharSet : 1, lfFaceName : "Segoe UI" };

HFONT getDialogFont()

{

if(winFont)

return winFont;

return newDialogFont();

}

int GetDesktopDPI()

{

HWND hwnd = GetDesktopWindow();

HDC hDDC = GetDC(hwnd);

int dpi = GetDeviceCaps(hDDC, LOGPIXELSY);

ReleaseDC(hwnd, hDDC);

return dpi;

}

HFONT newDialogFont()

{

// GetStockObject(DEFAULT\_GUI\_FONT);

//int nHeight = -MulDiv(dialogFontSize, GetDesktopDPI(), 72);

//winFont = CreateFontA(int cHeight, int cWidth, int cEscapement, int cOrientation, int cWeight, DWORD bItalic,

// DWORD bUnderline, DWORD bStrikeOut, DWORD iCharSet, DWORD iOutPrecision, DWORD iClipPrecision,

// DWORD iQuality, DWORD iPitchAndFamily, LPCSTR pszFaceName);

winFont = CreateFontIndirectW(&dialogLogFont);

assert(winFont);

return winFont;

}

HFONT deleteDialogFont(HFONT font)

{

if(font is null)

return null;

if(winFont is font)

winFont = null;

DeleteObject(font);

return null;

}

class Widget

{

HWND hwnd;

bool attached;

Widget parent;

Widget[] children;

this()

{

}

this(Widget p)

{

if(p)

p.addChild(this);

}

bool createWidget(Widget parent, string classname, string text, uint style, uint exstyle, int id)

{

HWND parenthwnd = parent ? parent.hwnd : null;

hwnd = CreateWindowExW(exstyle, toUTF16z(classname), toUTF16z(text), style,

CW\_USEDEFAULT, CW\_USEDEFAULT, 10, 10,

parenthwnd, cast(HMENU)id, hInst, null);

assert(hwnd !is null, "Failed to create " ~ classname ~ " window");

if(!hwnd)

return false;

SetWindowLongA(hwnd, GWL\_USERDATA, cast(int)cast(void\*)this);

return true;

}

void Dispose()

{

while(children.length)

{

Widget child = children[0];

child.Dispose();

delChild(child);

}

if(hwnd)

{

if(!attached)

{

BOOL ok = DestroyWindow(hwnd);

assert(ok);

}

hwnd = null;

}

}

void addChild(Widget child)

{

children ~= child;

child.parent = this;

}

void delChild(Widget child)

{

assert(child.parent is this);

for(int i = 0; i < children.length; i++)

if(children[i] is child)

{

children = children[0 .. i] ~ children[i+1 .. $];

child.parent = null;

break;

}

}

// coordinates relative to parent (child window) or screen (top level window)

bool getRect(ref int left, ref int top, ref int w, ref int h)

{

RECT r;

if(!.GetWindowRect(hwnd, &r))

return false;

if(HWND ph = GetParent(hwnd))

{

RECT pr;

if(!.GetWindowRect(ph, &pr))

return false;

r.left -= pr.left;

r.right -= pr.left;

r.top -= pr.top;

r.bottom -= pr.top;

}

left = r.left;

top = r.top;

w = r.right - r.left;

h = r.bottom - r.top;

return true;

}

// coordinates relative to parent (child window) or screen (top level window)

void setRect(int left, int top, int w, int h)

{

BOOL ok = MoveWindow(hwnd, left, top, w, h, true);

assert(ok, "Failed to move window in setRect");

}

void setVisible(bool visible)

{

ShowWindow(hwnd, visible ? SW\_SHOW : SW\_HIDE); // ignore bool result

}

void setEnabled(bool enable)

{

EnableWindow(hwnd, enable);

}

void SetFocus()

{

.SetFocus(hwnd);

}

void SetRedraw(bool enable)

{

SendMessage(WM\_SETREDRAW, enable);

}

int SendMessage(int msg, WPARAM wp = 0, LPARAM lp = 0)

{

return .SendMessage(hwnd, msg, wp, lp);

}

void InvalidateRect(RECT\* r, bool erase)

{

.InvalidateRect(hwnd, r, erase);

}

string GetWindowText()

{

WCHAR[256] txt;

int len = GetWindowTextW(hwnd, txt.ptr, txt.length);

if(len < txt.length)

return toUTF8(txt[0..len]);

scope buffer = new wchar[len+1];

len = GetWindowTextW(hwnd, buffer.ptr, len+1);

return toUTF8(buffer[0..len]);

}

bool SetWindowText(string txt)

{

return SetWindowTextW(hwnd, toUTF16z(txt)) != 0;

}

bool GetWindowRect(RECT\* r)

{

return .GetWindowRect(hwnd, r) != 0;

}

bool GetClientRect(RECT\* r)

{

return .GetClientRect(hwnd, r) != 0;

}

bool ScreenToClient(POINT \*lpPoint)

{

return .ScreenToClient(hwnd, lpPoint) != 0;

}

bool ScreenToClient(RECT \*rect)

{

POINT pnt = { rect.left, rect.top };

if (.ScreenToClient(hwnd, &pnt) == 0)

return false;

rect.right += pnt.x - rect.left;

rect.bottom += pnt.y - rect.top;

rect.left = pnt.x;

rect.top = pnt.y;

return true;

}

bool SetWindowPos(HWND hWndInsertAfter, int X, int Y, int cx, int cy, uint uFlags)

{

return .SetWindowPos(hwnd, hWndInsertAfter, X, Y, cx, cy, uFlags) != 0;

}

bool SetWindowPos(HWND hWndInsertAfter, RECT\* r, uint uFlags)

{

return .SetWindowPos(hwnd, hWndInsertAfter, r.left, r.top, r.right - r.left, r.bottom - r.top, uFlags) != 0;

}

bool SetWindowStyle(int style)

{

return SetWindowLongA(hwnd, GWL\_STYLE, style) != 0;

}

bool AddWindowStyle(int flag, int clear = 0)

{

DWORD style = GetWindowLongA(hwnd, GWL\_STYLE);

return SetWindowLongA(hwnd, GWL\_STYLE, (style & ~clear) | flag) != 0;

}

bool DelWindowStyle(int flag)

{

DWORD style = GetWindowLongA(hwnd, GWL\_STYLE);

return SetWindowLongA(hwnd, GWL\_STYLE, style & ~flag) != 0;

}

bool SetWindowExStyle(int style)

{

return SetWindowLongA(hwnd, GWL\_EXSTYLE, style) != 0;

}

bool AddWindowExStyle(int flag, int clear = 0)

{

DWORD style = GetWindowLongA(hwnd, GWL\_EXSTYLE);

return SetWindowLongA(hwnd, GWL\_EXSTYLE, (style & ~clear) | flag) != 0;

}

bool DelWindowExStyle(int flag)

{

DWORD style = GetWindowLongA(hwnd, GWL\_EXSTYLE);

return SetWindowLongA(hwnd, GWL\_EXSTYLE, style & ~flag) != 0;

}

static Widget fromHWND(HWND hwnd)

{

return cast(Widget)cast(void\*)GetWindowLongA(hwnd, GWL\_USERDATA);

}

static HINSTANCE getInstance() { return hInst; }

}

class Window : Widget

{

static bool hasRegistered = false;

static HBRUSH bgbrush;

static void registerClass()

{

if(hasRegistered)

return;

hasRegistered = true;

DWORD color = GetSysColor(COLOR\_BTNFACE);

bgbrush = CreateSolidBrush(color);

WNDCLASSA wc;

wc.lpszClassName = "VisualDWindow";

wc.style = CS\_OWNDC | CS\_HREDRAW | CS\_VREDRAW;

wc.lpfnWndProc = &WinWindowProc;

wc.hInstance = hInst;

wc.hIcon = null; //DefaultWindowIcon.peer;

//wc.hIconSm = DefaultWindowSmallIcon.peer;

wc.hCursor = LoadCursorW(cast(HINSTANCE) null, IDC\_ARROW);

wc.hbrBackground = bgbrush;

wc.lpszMenuName = null;

wc.cbClsExtra = 0;

wc.cbWndExtra = 0;

ATOM atom = RegisterClassA(&wc);

assert(atom);

}

static void unregisterClass()

{

if(!hasRegistered)

return;

hasRegistered = false;

UnregisterClassA("VisualDWindow", hInst);

if(bgbrush)

DeleteObject(bgbrush);

bgbrush = null;

}

this(in HWND h)

{

hwnd = cast(HWND) h; // we need to remove "const" from "in"

attached = true;

createdWindows[this] = this; // prevent garbage collection

}

this(Widget parent, string title = "", int id = 0)

{

registerClass();

uint style = WS\_VISIBLE;

if(parent)

style |= WS\_CHILD;

createWidget(parent, "VisualDWindow", title, style, 0, id);

createdWindows[this] = this; // prevent garbage collection

super(parent);

}

this(Widget parent, uint style, string title = "", int id = 0)

{

registerClass();

createWidget(parent, "VisualDWindow", title, style, 0, id);

createdWindows[this] = this; // prevent garbage collection

super(parent);

}

override void Dispose()

{

if(backgroundBrush)

DeleteObject(backgroundBrush);

super.Dispose();

createdWindows.remove(this);

}

void setBackground(DWORD col)

{

//if(backgroundBrush)

//        DeleteObject(backgroundBrush);

//backgroundBrush = CreateSolidBrush(col);

}

extern(Windows) static int WinWindowProc(HWND hWnd, uint uMsg, WPARAM wParam, LPARAM lParam)

{

if (Window win = cast(Window) fromHWND(hWnd))

return win.WindowProc(hWnd,uMsg,wParam,lParam);

return DefWindowProcA(hWnd, uMsg, wParam, lParam);

}

int WindowProc(HWND hWnd, uint uMsg, WPARAM wParam, LPARAM lParam)

{

switch (uMsg) {

case WM\_COMMAND:

Widget c = fromHWND(cast(HWND)lParam);

doCommand(c, LOWORD(wParam));

break;

case WM\_CTLCOLORBTN:

case WM\_CTLCOLORSTATIC:

HDC dc = cast(HDC)wParam;

//SetTextColor(dc, 0xFF0000);

SetBkColor(dc, GetSysColor(COLOR\_BTNFACE));

return cast(int)bgbrush;

case WM\_CLOSE:

// send close message to top level window

// otherwise, only our embedded window is closed when pressing esc with the focus in the multi-line-edit

if(HWND hnd = GetAncestor(hWnd, GA\_ROOT))

if(hnd != hwnd && hnd != hWnd)

return SendMessageA(hnd, uMsg, wParam, lParam);

break;

case WM\_DESTROY:

if(destroyDelegate)

destroyDelegate(this);

break;

case WM\_NOTIFY:

NMHDR\* hdr = cast(NMHDR\*) lParam;

if(applyDelegate)

if(hdr.code == PSN\_APPLY)

applyDelegate(this);

switch(hdr.code)

{

case TCN\_SELCHANGING:

// Return FALSE to allow the selection to change.

return FALSE;

case TCN\_SELCHANGE:

if(auto tc = cast(TabControl) fromHWND(hdr.hwndFrom))

tc.raiseWidget(tc.getCurSel());

return FALSE;

default:

break;

}

break;

default:

break;

}

return DefWindowProcA(hWnd, uMsg, wParam, lParam);

}

void delegate(Widget w, int cmd) commandDelegate;

void delegate(Widget w) destroyDelegate;

void delegate(Widget w) applyDelegate;

bool doCommand(Widget w, int cmd)

{

if(commandDelegate)

commandDelegate(w, cmd);

return true;

}

HANDLE backgroundBrush;

}

class Dialog : Widget

{

static bool hasRegistered = false;

static HBRUSH bgbrush;

static void registerClass()

{

if(hasRegistered)

return;

hasRegistered = true;

DWORD color = GetSysColor(COLOR\_BTNFACE);

bgbrush = CreateSolidBrush(color);

WNDCLASSA wc;

wc.lpszClassName = "VisualDDialog";

wc.style = CS\_DBLCLKS | CS\_SAVEBITS;

wc.lpfnWndProc = &DlgWindowProc;

wc.hInstance = hInst;

wc.hIcon = null; //DefaultWindowIcon.peer;

//wc.hIconSm = DefaultWindowSmallIcon.peer;

wc.hCursor = LoadCursorW(cast(HINSTANCE) null, IDC\_ARROW);

wc.hbrBackground = bgbrush;

wc.lpszMenuName = null;

wc.cbClsExtra = 0;

wc.cbWndExtra = DLGWINDOWEXTRA;

ATOM atom = RegisterClassA(&wc);

assert(atom);

}

static void unregisterClass()

{

if(!hasRegistered)

return;

hasRegistered = false;

UnregisterClassA("VisualDDialog", hInst);

if(bgbrush)

DeleteObject(bgbrush);

bgbrush = null;

}

extern(Windows) static int DlgWindowProc(HWND hWnd, uint uMsg, WPARAM wParam, LPARAM lParam)

{

if (Dialog dlg = cast(Dialog) fromHWND(hWnd))

return dlg.WindowProc(hWnd,uMsg,wParam,lParam);

return DefDlgProcA(hWnd, uMsg, wParam, lParam);

}

this(Widget parent, string text = "", int id = 0)

{

registerClass();

HWND parenthwnd = parent ? parent.hwnd : null; // VisualDDialog

createWidget(parent, "#32770", text, WS\_CHILD | WS\_VISIBLE | DS\_3DLOOK | DS\_CONTROL, 0, id);

SendMessageA(hwnd, WM\_SETFONT, cast(WPARAM)getDialogFont(), 0);

SetWindowLongA(hwnd, GWL\_WNDPROC, cast(int)cast(void\*)&DlgWindowProc);

super(parent);

}

int WindowProc(HWND hWnd, uint uMsg, WPARAM wParam, LPARAM lParam)

{

return DefDlgProcA(hWnd, uMsg, wParam, lParam);

}

}

class Label : Widget

{

this(Widget parent, string text = "", int id = 0)

{

HWND parenthwnd = parent ? parent.hwnd : null;

createWidget(parent, "STATIC", text, SS\_LEFTNOWORDWRAP | WS\_CHILD | WS\_VISIBLE, 0, id);

SendMessageA(hwnd, WM\_SETFONT, cast(WPARAM)        getDialogFont(), 0);

super(parent);

}

}

class Text : Widget

{

this(Widget parent, string text = "", int id = 0)

{

this(parent, text, id, ES\_AUTOHSCROLL, WS\_EX\_STATICEDGE);

}

this(Widget parent, string text, int id, int style, int exstyle)

{

HWND parenthwnd = parent ? parent.hwnd : null;

createWidget(parent, "EDIT", text, style | WS\_CHILD | WS\_VISIBLE | WS\_TABSTOP, exstyle, id);

SendMessageA(hwnd, WM\_SETFONT, cast(WPARAM)getDialogFont(), 0);

super(parent);

}

void setText(string str)

{

auto lines = std.string.splitLines(str);

string newline = std.ascii.newline; // join no longer likes immutable seperator

auto winstr = std.string.join(lines, newline);

SendMessageW(hwnd, WM\_SETTEXT, 0, cast(LPARAM)toUTF16z(winstr));

}

void setText(wstring str)

{

auto lines = std.string.splitLines(str);

static if(\_\_traits(compiles, std.string.join(lines, "\r\n")))

auto winstr = std.string.join(lines, "\r\n") ~ "\0";

else

{

wstring winstr;

if(lines.length > 0)

winstr = lines[0];

for(int i = 1; i < lines.length; i++)

winstr ~= "\r\n" ~ lines[i];

winstr ~= "\0";

}

SendMessageW(hwnd, WM\_SETTEXT, 0, cast(LPARAM)winstr.ptr);

}

string getText()

{

int len = SendMessageW(hwnd, WM\_GETTEXTLENGTH, 0, 0);

scope buffer = new wchar[len+1];

SendMessageW(hwnd, WM\_GETTEXT, cast(WPARAM)(len+1), cast(LPARAM)buffer.ptr);

string s = toUTF8(buffer[0..$-1]);

s = replace(s, "\r", "");

return s;

}

wstring getWText()

{

int len = SendMessageW(hwnd, WM\_GETTEXTLENGTH, 0, 0);

auto buffer = new wchar[len+1];

SendMessageW(hwnd, WM\_GETTEXT, cast(WPARAM)(len+1), cast(LPARAM)buffer.ptr);

buffer = replace(buffer, "\r", "");

return assumeUnique(buffer[0..$-1]);

}

}

class MultiLineText : Text

{

this(Widget parent, string text = "", int id = 0, bool readonly = false)

{

scope lines = std.string.splitLines(text);

string newline = std.ascii.newline;

scope winstr = std.string.join(lines, newline);

uint exstyle = /\*WS\_HSCROLL |\*/ WS\_VSCROLL | ES\_WANTRETURN | ES\_MULTILINE | ES\_AUTOVSCROLL | ES\_AUTOHSCROLL;

if(readonly)

exstyle = (exstyle & ~(WS\_HSCROLL | ES\_AUTOHSCROLL)) | ES\_READONLY;

super(parent, winstr, id, exstyle, 0);

defWndProc = cast(WNDPROC)cast(void\*)GetWindowLongA(hwnd, GWL\_WNDPROC);

SetWindowLongA(hwnd, GWL\_WNDPROC, cast(int)cast(void\*)&MLTWindowProc);

}

extern(Windows) static int MLTWindowProc(HWND hWnd, uint uMsg, WPARAM wParam, LPARAM lParam)

{

if (MultiLineText mlt = cast(MultiLineText) fromHWND(hWnd))

return mlt.WindowProc(hWnd,uMsg,wParam,lParam);

return DefWindowProcA(hWnd, uMsg, wParam, lParam);

}

WNDPROC defWndProc;

static HWND FindDialog(HWND hWnd)

{

while(hWnd && (GetWindowLongA(hWnd, GWL\_STYLE) & (WS\_POPUP | WS\_SYSMENU)) == 0)

hWnd = GetParent(hWnd);

return hWnd;

}

int WindowProc(HWND hWnd, uint uMsg, WPARAM wParam, LPARAM lParam)

{

if(uMsg == WM\_CHAR)

{

switch(wParam)

{

case VK\_TAB:

bool shift = (0x80 & GetKeyState(VK\_SHIFT)) != 0;

.SetFocus(GetNextDlgTabItem(FindDialog(hWnd), hWnd, shift));

break;

default:

break;

}

}

//return DefWindowProcA(hWnd, uMsg, wParam, lParam);

return CallWindowProcA(cast(FARPROC) defWndProc, hWnd, uMsg, wParam, lParam);

}

}

class ComboBox : Widget

{

this(Widget parent, string[] texts, bool editable = true, int id = 0)

{

HWND parenthwnd = parent ? parent.hwnd : null;

DWORD style = editable ? CBS\_DROPDOWN | CBS\_AUTOHSCROLL : CBS\_DROPDOWNLIST;

createWidget(parent, "COMBOBOX", "", style | WS\_VSCROLL | WS\_HSCROLL | WS\_CHILD | WS\_VISIBLE | WS\_TABSTOP, 0, id);

SendMessageA(hwnd, WM\_SETFONT, cast(WPARAM)getDialogFont(), 0);

foreach (s; texts)

SendMessageW(hwnd, CB\_ADDSTRING, 0, cast(LPARAM)toUTF16z(s));

super(parent);

}

int findString(string s)

{

return SendMessageW(hwnd, CB\_FINDSTRING, 0, cast(LPARAM)toUTF16z(s));

}

int getSelection()

{

return SendMessageA(hwnd, CB\_GETCURSEL, 0, 0);

}

void setSelection(int n)

{

SendMessageA(hwnd, CB\_SETCURSEL, n, 0);

}

void setSelection(string s)

{

SendMessageA(hwnd, CB\_SELECTSTRING, 0, cast(LPARAM)toUTF16z(s));

}

string getText()

{

int len = SendMessageW(hwnd, WM\_GETTEXTLENGTH, 0, 0);

scope buffer = new wchar[len+1];

SendMessageW(hwnd, WM\_GETTEXT, cast(WPARAM)(len+1), cast(LPARAM)buffer.ptr);

return toUTF8(buffer[0..$-1]);

}

wstring getWText()

{

int len = SendMessageW(hwnd, WM\_GETTEXTLENGTH, 0, 0);

scope buffer = new wchar[len+1];

SendMessageW(hwnd, WM\_GETTEXT, cast(WPARAM)(len+1), cast(LPARAM)buffer.ptr);

return assumeUnique(buffer[0..$-1]);

}

}

class ButtonBase : Widget

{

this(Widget parent) { super(parent); }

bool isChecked()

{

bool res = SendMessageA(hwnd, BM\_GETCHECK, 0, 0) == BST\_CHECKED;

return res;

}

void setChecked(bool x)

{

SendMessageA(hwnd, BM\_SETCHECK, x ? BST\_CHECKED : BST\_UNCHECKED, 0);

}

}

class CheckBox : ButtonBase

{

this(Widget parent, string intext, int id = 0)

{

HWND parenthwnd = parent ? parent.hwnd : null;

createWidget(parent, "BUTTON", intext, BS\_AUTOCHECKBOX | WS\_CHILD | WS\_VISIBLE | WS\_TABSTOP, 0, id);

SendMessageA(hwnd, WM\_SETFONT, cast(WPARAM)getDialogFont(), 0);

super(parent);

}

}

class Button : ButtonBase

{

this(Widget parent, string intext, int id = 0)

{

HWND parenthwnd = parent ? parent.hwnd : null;

createWidget(parent, "BUTTON", intext, BS\_PUSHBUTTON | WS\_CHILD | WS\_VISIBLE | WS\_TABSTOP, 0, id);

SendMessageA(hwnd, WM\_SETFONT, cast(WPARAM)getDialogFont(), 0);

super(parent);

}

}

class Frame : ButtonBase

{

this(Widget parent, string intext = "", int id = 0)

{

HWND parenthwnd = parent ? parent.hwnd : null;

createWidget(parent, "BUTTON", intext, BS\_GROUPBOX | WS\_CHILD | WS\_VISIBLE, 0, id);

SendMessageA(hwnd, WM\_SETFONT, cast(WPARAM)getDialogFont(), 0);

super(parent);

}

}

class ToolBar : Widget

{

this(Widget parent, uint style, uint exstyle, int id = 0)

{

HWND parenthwnd = parent ? parent.hwnd : null;

createWidget(parent, TOOLBARCLASSNAMEA, "", style | WS\_CHILD | WS\_VISIBLE, exstyle, id);

super(parent);

}

bool EnableCheckButton(uint id, bool enable, bool check)

{

TBBUTTONINFO tbi;

tbi.cbSize = TBBUTTONINFO.sizeof;

tbi.dwMask = TBIF\_STATE;

tbi.fsState = (enable ? TBSTATE\_ENABLED : 0)

| (check ? TBSTATE\_CHECKED : 0);

return .SendMessage(hwnd, TB\_SETBUTTONINFO, id, cast(LPARAM)&tbi) != 0;

}

}

class ListView : Widget

{

this(Widget parent, uint style, uint exstyle, int id = 0)

{

HWND parenthwnd = parent ? parent.hwnd : null;

createWidget(parent, "SysListView32", "", style | WS\_CHILD | WS\_VISIBLE | WS\_TABSTOP, exstyle, id);

super(parent);

}

int SendItemMessage(uint msg, ref LVITEM lvi)

{

return .SendMessage(hwnd, msg, 0, cast(LPARAM)&lvi);

}

}

class TabControl : Widget

{

this(Widget parent, string[] tabs, uint style = 0, uint exstyle = 0, int id = 0)

{

HWND parenthwnd = parent ? parent.hwnd : null;

createWidget(parent, "SysTabControl32", "", style | WS\_CHILD | WS\_VISIBLE | WS\_TABSTOP, exstyle, id);

SendMessageA(hwnd, WM\_SETFONT, cast(WPARAM)getDialogFont(), 0);

super(parent);

foreach(i, t; tabs)

{

TCITEM item;

item.mask = TCIF\_TEXT;

item.iImage = -1;

item.pszText = cast(wchar\*)toUTF16z(t);

SendMessageW(hwnd, TCM\_INSERTITEMW, i, cast(LPARAM)&item);

auto p = new Window(parent);

pages ~= p;

}

raiseWidget(0);

}

override bool SetWindowPos(HWND hWndInsertAfter, int X, int Y, int cx, int cy, uint uFlags)

{

if(!super.SetWindowPos(hWndInsertAfter, X, Y, cx, cy, uFlags))

return false;

if(uFlags & SWP\_NOSIZE)

return false;

setPageSize(X, Y, cx, cy);

return true;

}

void setPageSize(int X, int Y, int cx, int cy)

{

RECT r;

r.left = X;

r.right = X + cx;

r.top = Y;

r.bottom = Y + cy;

SendMessage(TCM\_ADJUSTRECT, false, cast(LPARAM)&r);

foreach(p; pages)

p.SetWindowPos(null, &r, SWP\_NOZORDER | SWP\_NOACTIVATE);

}

void setHeaderSize(int X, int Y, int cx, int cy)

{

RECT r;

r.left = X;

r.right = X + cx;

r.top = Y;

r.bottom = Y + cy;

SendMessage(TCM\_ADJUSTRECT, false, cast(LPARAM)&r);

super.setRect(X, Y, cx, r.top - Y);

}

override void setRect(int left, int top, int w, int h)

{

setHeaderSize(left, top, w, h);

setPageSize(left, top, w, h);

}

override bool GetWindowRect(RECT\* rect)

{

if(!super.GetWindowRect(rect))

return false;

if(!pages.empty)

{

RECT pr;

if (!pages[0].GetWindowRect(&pr))

return false;

rect.bottom = pr.bottom;

}

return true;

}

// space for header and footer

int getFrameHeight()

{

RECT r;

r.left = 0;

r.right = 100;

r.top = 0;

r.bottom = 100;

SendMessage(TCM\_ADJUSTRECT, false, cast(LPARAM)&r);

return r.top + (100 - r.bottom);

}

int getCurSel()

{

return SendMessage(TCM\_GETCURSEL, 0, 0);

}

void raiseWidget(size\_t idx)

{

foreach(i, p; pages)

p.setVisible(i == idx);

}

Widget[] pages;

}

int PopupContextMenu(HWND hwnd, POINT pt, wstring[] entries, int check = -1, int presel = -1)

{

HMENU hmnu = CreatePopupMenu();

if(!hmnu)

return -1;

scope(exit) DestroyMenu(hmnu);

MENUITEMINFO mii;

mii.cbSize = mii.sizeof;

mii.fMask = MIIM\_FTYPE | MIIM\_ID | MIIM\_STATE | MIIM\_STRING;

mii.fType = MFT\_STRING;

wchar\*[] entriesz;

for (int i = 0; i < entries.length; i++)

{

mii.fState = (i == check ? MFS\_CHECKED : 0) | (i == presel ? MFS\_DEFAULT : 0);

wchar\* pz = cast(wchar\*) (entries[i] ~ '\0').ptr;

entriesz ~= pz;

mii.wID = i + 1;

mii.dwTypeData = pz;

if(!InsertMenuItem(hmnu, cast(UINT)i + 1, TRUE, &mii))

return -1;

}

UINT uiCmd = TrackPopupMenuEx(hmnu, TPM\_RETURNCMD | TPM\_NONOTIFY | TPM\_HORIZONTAL | TPM\_TOPALIGN | TPM\_LEFTALIGN, pt.x, pt.y, hwnd, null);

if (uiCmd)

return uiCmd - 1;

HRESULT hr = HResultFromLastError();

return -1;

}

struct Attachment

{

// specify the fraction that the control receives from a size change

short hdiv;

short left; // left edge will receive left/hdiv of change

short right;

short vdiv;

short top;

short bottom;

}

enum kAttachNone = Attachment(1, 0, 0, 1, 0, 0);

enum kAttachLeftRight = Attachment(1, 0, 1, 1, 0, 0);

enum kAttachRight = Attachment(1, 1, 1, 1, 0, 0);

enum kAttachTopBottom = Attachment(1, 0, 0, 1, 0, 1);

enum kAttachBottom = Attachment(1, 0, 0, 1, 1, 1);

enum kAttachAll = Attachment(1, 0, 1, 1, 0, 1);

struct AttachData

{

Attachment att;

short initleft; // initial rect of child window relative to parent

short initright;

short inittop;

short initbottom;

short initwidth; // initial parent width

short initheight; // initial parent height

bool initFromWidget(Widget w)

{

RECT r, pr;

if (!w.GetWindowRect(&r))

return false;

if (!w.parent || !w.parent.GetWindowRect(&pr))

return false;

initwidth = cast(short) (pr.right - pr.left);

initheight = cast(short) (pr.bottom - pr.top);

initleft = cast(short) (r.left - pr.left);

initright = cast(short) (r.right - pr.left);

inittop = cast(short) (r.top - pr.top);

initbottom = cast(short) (r.bottom - pr.top);

//logCall("initFromWidget(", w, ":", cast(void\*)w, ") = w:", initwidth, " h:", initheight, " l:", initleft, " r:", initright, " t:", inittop, " b:", initbottom);

return true;

}

bool resizeWidget(Widget w)

{

RECT pr;

if (!w.parent || !w.parent.GetWindowRect(&pr))

return false;

int dx = pr.right - pr.left - initwidth;

int dy = pr.bottom - pr.top - initheight;

int hdiv = max(1, att.hdiv);

int vdiv = max(1, att.vdiv);

int nleft = initleft + dx \* att.left / hdiv;

int nright = initright + dx \* att.right / hdiv;

int ntop = inittop + dy \* att.top / vdiv;

int nbottom = initbottom + dy \* att.bottom / vdiv;

//logCall("resizeWidget(", w, ":", cast(void\*)w, ") to [l:", nleft, " t:", ntop, " w:", nright - nleft, " h:", nbottom - ntop, "]");

w.setRect(nleft, ntop, nright - nleft, nbottom - ntop);

return true;

}

}

bool initWinControls(HINSTANCE inst)

{

hInst = inst;

Window.registerClass();

Dialog.registerClass();

return true;

}

bool exitWinControls(HINSTANCE inst)

{

Window.unregisterClass();

Dialog.unregisterClass();

return true;

}

// This file is part of Visual D

//

// Visual D integrates the D programming language into Visual Studio

// Copyright (c) 2010 by Rainer Schuetze, All Rights Reserved

//

// Distributed under the Boost Software License, Version 1.0.

// See accompanying file LICENSE\_1\_0.txt or copy at <http://www.boost.org/LICENSE_1_0.txt>

module visuald.windows;

import sdk.win32.commctrl;

HRESULT HResultFromLastError()

{

return HRESULT\_FROM\_WIN32(GetLastError());

}

int GET\_X\_LPARAM(LPARAM lp)

{

return cast(int)cast(short)LOWORD(lp);

}

int GET\_Y\_LPARAM(LPARAM lp)

{

return cast(int)cast(short)HIWORD(lp);

}

int MAKELPARAM(int lo, int hi)

{

return (lo & 0xffff) | (hi << 16);

}

COLORREF RGB(int r, int g, int b)

{

return cast(COLORREF)(cast(BYTE)r | ((cast(uint)cast(BYTE)g)<<8) | ((cast(uint)cast(BYTE)b)<<16));

}

public import sdk.win32.shellapi;

const WM\_SYSTIMER = 0x118;

public import sdk.port.base;

extern(Windows)

{

uint GetThreadLocale();

UINT DragQueryFileW(HANDLE hDrop, UINT iFile, LPWSTR lpszFile, UINT cch);

HINSTANCE ShellExecuteW(HWND hwnd, LPCWSTR lpOperation, LPCWSTR lpFile, LPCWSTR lpParameters, LPCWSTR lpDirectory, INT nShowCmd);

}

// use instead of ImageList\_LoadImage to avoid reduction to 16 color bitmaps

HIMAGELIST LoadImageList(HINSTANCE hi, LPCSTR lpbmp, int cx, int cy)

{

auto imglist = ImageList\_Create(cx, cy, ILC\_MASK | ILC\_COLOR24, cx \* 10, cx \* 10);

if(!imglist)

return null;

auto img = LoadImageA(hi, lpbmp, IMAGE\_BITMAP, 0, 0, LR\_LOADTRANSPARENT);

if(!img)

{

ImageList\_Destroy(imglist);

return null;

}

ImageList\_AddMasked(imglist, img, CLR\_DEFAULT);

DeleteObject(img);

return imglist;

}

// This file is part of Visual D

//

// Visual D integrates the D programming language into Visual Studio

// Copyright (c) 2010 by Rainer Schuetze, All Rights Reserved

//

// Distributed under the Boost Software License, Version 1.0.

// See accompanying file LICENSE\_1\_0.txt or copy at <http://www.boost.org/LICENSE_1_0.txt>

module visuald.wmmsg;

import visuald.windows;

import visuald.logutil;

string msg\_toString(uint msg)

{

if(msg == WM\_NULL) return "WM\_NULL";

if(msg == WM\_CREATE) return "WM\_CREATE";

if(msg == WM\_DESTROY) return "WM\_DESTROY";

if(msg == WM\_MOVE) return "WM\_MOVE";

if(msg == WM\_SIZE) return "WM\_SIZE";

if(msg == WM\_ACTIVATE) return "WM\_ACTIVATE";

if(msg == WM\_SETFOCUS) return "WM\_SETFOCUS";

if(msg == WM\_KILLFOCUS) return "WM\_KILLFOCUS";

if(msg == WM\_ENABLE) return "WM\_ENABLE";

if(msg == WM\_SETREDRAW) return "WM\_SETREDRAW";

if(msg == WM\_SETTEXT) return "WM\_SETTEXT";

if(msg == WM\_GETTEXT) return "WM\_GETTEXT";

if(msg == WM\_GETTEXTLENGTH) return "WM\_GETTEXTLENGTH";

if(msg == WM\_PAINT) return "WM\_PAINT";

if(msg == WM\_CLOSE) return "WM\_CLOSE";

if(msg == WM\_QUERYENDSESSION) return "WM\_QUERYENDSESSION";

if(msg == WM\_QUERYOPEN) return "WM\_QUERYOPEN";

if(msg == WM\_ENDSESSION) return "WM\_ENDSESSION";

if(msg == WM\_QUIT) return "WM\_QUIT";

if(msg == WM\_ERASEBKGND) return "WM\_ERASEBKGND";

if(msg == WM\_SYSCOLORCHANGE) return "WM\_SYSCOLORCHANGE";

if(msg == WM\_SHOWWINDOW) return "WM\_SHOWWINDOW";

if(msg == WM\_WININICHANGE) return "WM\_WININICHANGE";

if(msg == WM\_WININICHANGE ) return "WM\_WININICHANGE ";

if(msg == WM\_DEVMODECHANGE) return "WM\_DEVMODECHANGE";

if(msg == WM\_ACTIVATEAPP) return "WM\_ACTIVATEAPP";

if(msg == WM\_FONTCHANGE) return "WM\_FONTCHANGE";

if(msg == WM\_TIMECHANGE) return "WM\_TIMECHANGE";

if(msg == WM\_CANCELMODE) return "WM\_CANCELMODE";

if(msg == WM\_SETCURSOR) return "WM\_SETCURSOR";

if(msg == WM\_MOUSEACTIVATE) return "WM\_MOUSEACTIVATE";

if(msg == WM\_CHILDACTIVATE) return "WM\_CHILDACTIVATE";

if(msg == WM\_QUEUESYNC) return "WM\_QUEUESYNC";

if(msg == WM\_GETMINMAXINFO) return "WM\_GETMINMAXINFO";

if(msg == WM\_PAINTICON) return "WM\_PAINTICON";

if(msg == WM\_ICONERASEBKGND) return "WM\_ICONERASEBKGND";

if(msg == WM\_NEXTDLGCTL) return "WM\_NEXTDLGCTL";

if(msg == WM\_SPOOLERSTATUS) return "WM\_SPOOLERSTATUS";

if(msg == WM\_DRAWITEM) return "WM\_DRAWITEM";

if(msg == WM\_MEASUREITEM) return "WM\_MEASUREITEM";

if(msg == WM\_DELETEITEM) return "WM\_DELETEITEM";

if(msg == WM\_VKEYTOITEM) return "WM\_VKEYTOITEM";

if(msg == WM\_CHARTOITEM) return "WM\_CHARTOITEM";

if(msg == WM\_SETFONT) return "WM\_SETFONT";

if(msg == WM\_GETFONT) return "WM\_GETFONT";

if(msg == WM\_SETHOTKEY) return "WM\_SETHOTKEY";

if(msg == WM\_GETHOTKEY) return "WM\_GETHOTKEY";

if(msg == WM\_QUERYDRAGICON) return "WM\_QUERYDRAGICON";

if(msg == WM\_COMPAREITEM) return "WM\_COMPAREITEM";

if(msg == WM\_GETOBJECT) return "WM\_GETOBJECT";

if(msg == WM\_COMPACTING) return "WM\_COMPACTING";

if(msg == WM\_COMMNOTIFY) return "WM\_COMMNOTIFY";

if(msg == WM\_WINDOWPOSCHANGING) return "WM\_WINDOWPOSCHANGING";

if(msg == WM\_WINDOWPOSCHANGED) return "WM\_WINDOWPOSCHANGED";

if(msg == WM\_POWER) return "WM\_POWER";

if(msg == WM\_NOTIFY) return "WM\_NOTIFY";

if(msg == WM\_INPUTLANGCHANGEREQUEST) return "WM\_INPUTLANGCHANGEREQUEST";

if(msg == WM\_INPUTLANGCHANGE) return "WM\_INPUTLANGCHANGE";

if(msg == WM\_TCARD) return "WM\_TCARD";

if(msg == WM\_HELP) return "WM\_HELP";

if(msg == WM\_USERCHANGED) return "WM\_USERCHANGED";

if(msg == WM\_NOTIFYFORMAT) return "WM\_NOTIFYFORMAT";

if(msg == WM\_CONTEXTMENU) return "WM\_CONTEXTMENU";

if(msg == WM\_STYLECHANGING) return "WM\_STYLECHANGING";

if(msg == WM\_STYLECHANGED) return "WM\_STYLECHANGED";

if(msg == WM\_DISPLAYCHANGE) return "WM\_DISPLAYCHANGE";

if(msg == WM\_GETICON) return "WM\_GETICON";

if(msg == WM\_SETICON) return "WM\_SETICON";

if(msg == WM\_NCCREATE) return "WM\_NCCREATE";

if(msg == WM\_NCDESTROY) return "WM\_NCDESTROY";

if(msg == WM\_NCCALCSIZE) return "WM\_NCCALCSIZE";

if(msg == WM\_NCHITTEST) return "WM\_NCHITTEST";

if(msg == WM\_NCPAINT) return "WM\_NCPAINT";

if(msg == WM\_NCACTIVATE) return "WM\_NCACTIVATE";

if(msg == WM\_GETDLGCODE) return "WM\_GETDLGCODE";

if(msg == WM\_SYNCPAINT) return "WM\_SYNCPAINT";

if(msg == WM\_NCMOUSEMOVE) return "WM\_NCMOUSEMOVE";

if(msg == WM\_NCLBUTTONDOWN) return "WM\_NCLBUTTONDOWN";

if(msg == WM\_NCLBUTTONUP) return "WM\_NCLBUTTONUP";

if(msg == WM\_NCLBUTTONDBLCLK) return "WM\_NCLBUTTONDBLCLK";

if(msg == WM\_NCRBUTTONDOWN) return "WM\_NCRBUTTONDOWN";

if(msg == WM\_NCRBUTTONUP) return "WM\_NCRBUTTONUP";

if(msg == WM\_NCRBUTTONDBLCLK) return "WM\_NCRBUTTONDBLCLK";

if(msg == WM\_NCMBUTTONDOWN) return "WM\_NCMBUTTONDOWN";

if(msg == WM\_NCMBUTTONUP) return "WM\_NCMBUTTONUP";

if(msg == WM\_NCMBUTTONDBLCLK) return "WM\_NCMBUTTONDBLCLK";

if(msg == WM\_NCXBUTTONDOWN) return "WM\_NCXBUTTONDOWN";

if(msg == WM\_NCXBUTTONUP) return "WM\_NCXBUTTONUP";

if(msg == WM\_NCXBUTTONDBLCLK) return "WM\_NCXBUTTONDBLCLK";

if(msg == WM\_INPUT\_DEVICE\_CHANGE) return "WM\_INPUT\_DEVICE\_CHANGE";

if(msg == WM\_INPUT) return "WM\_INPUT";

if(msg == WM\_KEYFIRST) return "WM\_KEYFIRST";

if(msg == WM\_KEYDOWN) return "WM\_KEYDOWN";

if(msg == WM\_KEYUP) return "WM\_KEYUP";

if(msg == WM\_CHAR) return "WM\_CHAR";

if(msg == WM\_DEADCHAR) return "WM\_DEADCHAR";

if(msg == WM\_SYSKEYDOWN) return "WM\_SYSKEYDOWN";

if(msg == WM\_SYSKEYUP) return "WM\_SYSKEYUP";

if(msg == WM\_SYSCHAR) return "WM\_SYSCHAR";

if(msg == WM\_SYSDEADCHAR) return "WM\_SYSDEADCHAR";

if(msg == WM\_UNICHAR) return "WM\_UNICHAR";

if(msg == WM\_KEYLAST) return "WM\_KEYLAST";

if(msg == UNICODE\_NOCHAR) return "UNICODE\_NOCHAR";

if(msg == WM\_IME\_STARTCOMPOSITION) return "WM\_IME\_STARTCOMPOSITION";

if(msg == WM\_IME\_ENDCOMPOSITION) return "WM\_IME\_ENDCOMPOSITION";

if(msg == WM\_IME\_COMPOSITION) return "WM\_IME\_COMPOSITION";

if(msg == WM\_IME\_KEYLAST) return "WM\_IME\_KEYLAST";

if(msg == WM\_INITDIALOG) return "WM\_INITDIALOG";

if(msg == WM\_COMMAND) return "WM\_COMMAND";

if(msg == WM\_SYSCOMMAND) return "WM\_SYSCOMMAND";

if(msg == WM\_TIMER) return "WM\_TIMER";

if(msg == WM\_HSCROLL) return "WM\_HSCROLL";

if(msg == WM\_VSCROLL) return "WM\_VSCROLL";

if(msg == WM\_INITMENU) return "WM\_INITMENU";

if(msg == WM\_INITMENUPOPUP) return "WM\_INITMENUPOPUP";

if(msg == WM\_MENUSELECT) return "WM\_MENUSELECT";

if(msg == WM\_MENUCHAR) return "WM\_MENUCHAR";

if(msg == WM\_ENTERIDLE) return "WM\_ENTERIDLE";

if(msg == WM\_MENURBUTTONUP) return "WM\_MENURBUTTONUP";

if(msg == WM\_MENUDRAG) return "WM\_MENUDRAG";

if(msg == WM\_MENUGETOBJECT) return "WM\_MENUGETOBJECT";

if(msg == WM\_UNINITMENUPOPUP) return "WM\_UNINITMENUPOPUP";

if(msg == WM\_MENUCOMMAND) return "WM\_MENUCOMMAND";

if(msg == WM\_CHANGEUISTATE) return "WM\_CHANGEUISTATE";

if(msg == WM\_UPDATEUISTATE) return "WM\_UPDATEUISTATE";

if(msg == WM\_QUERYUISTATE) return "WM\_QUERYUISTATE";

if(msg == WM\_CTLCOLORMSGBOX) return "WM\_CTLCOLORMSGBOX";

if(msg == WM\_CTLCOLOREDIT) return "WM\_CTLCOLOREDIT";

if(msg == WM\_CTLCOLORLISTBOX) return "WM\_CTLCOLORLISTBOX";

if(msg == WM\_CTLCOLORBTN) return "WM\_CTLCOLORBTN";

if(msg == WM\_CTLCOLORDLG) return "WM\_CTLCOLORDLG";

if(msg == WM\_CTLCOLORSCROLLBAR) return "WM\_CTLCOLORSCROLLBAR";

if(msg == WM\_CTLCOLORSTATIC) return "WM\_CTLCOLORSTATIC";

if(msg == MN\_GETHMENU) return "MN\_GETHMENU";

if(msg == WM\_MOUSEMOVE) return "WM\_MOUSEMOVE";

if(msg == WM\_LBUTTONDOWN) return "WM\_LBUTTONDOWN";

if(msg == WM\_LBUTTONUP) return "WM\_LBUTTONUP";

if(msg == WM\_LBUTTONDBLCLK) return "WM\_LBUTTONDBLCLK";

if(msg == WM\_RBUTTONDOWN) return "WM\_RBUTTONDOWN";

if(msg == WM\_RBUTTONUP) return "WM\_RBUTTONUP";

if(msg == WM\_RBUTTONDBLCLK) return "WM\_RBUTTONDBLCLK";

if(msg == WM\_MBUTTONDOWN) return "WM\_MBUTTONDOWN";

if(msg == WM\_MBUTTONUP) return "WM\_MBUTTONUP";

if(msg == WM\_MBUTTONDBLCLK) return "WM\_MBUTTONDBLCLK";

if(msg == WM\_MOUSEWHEEL) return "WM\_MOUSEWHEEL";

if(msg == WM\_XBUTTONDOWN) return "WM\_XBUTTONDOWN";

if(msg == WM\_XBUTTONUP) return "WM\_XBUTTONUP";

if(msg == WM\_XBUTTONDBLCLK) return "WM\_XBUTTONDBLCLK";

if(msg == WM\_MOUSEHWHEEL) return "WM\_MOUSEHWHEEL";

if(msg == WM\_PARENTNOTIFY) return "WM\_PARENTNOTIFY";

if(msg == WM\_ENTERMENULOOP) return "WM\_ENTERMENULOOP";

if(msg == WM\_EXITMENULOOP) return "WM\_EXITMENULOOP";

if(msg == WM\_NEXTMENU) return "WM\_NEXTMENU";

if(msg == WM\_SIZING) return "WM\_SIZING";

if(msg == WM\_CAPTURECHANGED) return "WM\_CAPTURECHANGED";

if(msg == WM\_MOVING) return "WM\_MOVING";

if(msg == WM\_POWERBROADCAST) return "WM\_POWERBROADCAST";

if(msg == WM\_DEVICECHANGE) return "WM\_DEVICECHANGE";

if(msg == WM\_MDICREATE) return "WM\_MDICREATE";

if(msg == WM\_MDIDESTROY) return "WM\_MDIDESTROY";

if(msg == WM\_MDIACTIVATE) return "WM\_MDIACTIVATE";

if(msg == WM\_MDIRESTORE) return "WM\_MDIRESTORE";

if(msg == WM\_MDINEXT) return "WM\_MDINEXT";

if(msg == WM\_MDIMAXIMIZE) return "WM\_MDIMAXIMIZE";

if(msg == WM\_MDITILE) return "WM\_MDITILE";

if(msg == WM\_MDICASCADE) return "WM\_MDICASCADE";

if(msg == WM\_MDIICONARRANGE) return "WM\_MDIICONARRANGE";

if(msg == WM\_MDIGETACTIVE) return "WM\_MDIGETACTIVE";

if(msg == WM\_MDISETMENU) return "WM\_MDISETMENU";

if(msg == WM\_ENTERSIZEMOVE) return "WM\_ENTERSIZEMOVE";

if(msg == WM\_EXITSIZEMOVE) return "WM\_EXITSIZEMOVE";

if(msg == WM\_DROPFILES) return "WM\_DROPFILES";

if(msg == WM\_MDIREFRESHMENU) return "WM\_MDIREFRESHMENU";

if(msg == WM\_IME\_SETCONTEXT) return "WM\_IME\_SETCONTEXT";

if(msg == WM\_IME\_NOTIFY) return "WM\_IME\_NOTIFY";

if(msg == WM\_IME\_CONTROL) return "WM\_IME\_CONTROL";

if(msg == WM\_IME\_COMPOSITIONFULL) return "WM\_IME\_COMPOSITIONFULL";

if(msg == WM\_IME\_SELECT) return "WM\_IME\_SELECT";

if(msg == WM\_IME\_CHAR) return "WM\_IME\_CHAR";

if(msg == WM\_IME\_REQUEST) return "WM\_IME\_REQUEST";

if(msg == WM\_IME\_KEYDOWN) return "WM\_IME\_KEYDOWN";

if(msg == WM\_IME\_KEYUP) return "WM\_IME\_KEYUP";

if(msg == WM\_MOUSEHOVER) return "WM\_MOUSEHOVER";

if(msg == WM\_MOUSELEAVE) return "WM\_MOUSELEAVE";

if(msg == WM\_NCMOUSEHOVER) return "WM\_NCMOUSEHOVER";

if(msg == WM\_NCMOUSELEAVE) return "WM\_NCMOUSELEAVE";

if(msg == WM\_WTSSESSION\_CHANGE) return "WM\_WTSSESSION\_CHANGE";

if(msg >= WM\_TABLET\_FIRST && msg <= WM\_TABLET\_LAST) return "WM\_TABLET\_nnn";

if(msg == WM\_CUT) return "WM\_CUT";

if(msg == WM\_COPY) return "WM\_COPY";

if(msg == WM\_PASTE) return "WM\_PASTE";

if(msg == WM\_CLEAR) return "WM\_CLEAR";

if(msg == WM\_UNDO) return "WM\_UNDO";

if(msg == WM\_RENDERFORMAT) return "WM\_RENDERFORMAT";

if(msg == WM\_RENDERALLFORMATS) return "WM\_RENDERALLFORMATS";

if(msg == WM\_DESTROYCLIPBOARD) return "WM\_DESTROYCLIPBOARD";

if(msg == WM\_DRAWCLIPBOARD) return "WM\_DRAWCLIPBOARD";

if(msg == WM\_PAINTCLIPBOARD) return "WM\_PAINTCLIPBOARD";

if(msg == WM\_VSCROLLCLIPBOARD) return "WM\_VSCROLLCLIPBOARD";

if(msg == WM\_SIZECLIPBOARD) return "WM\_SIZECLIPBOARD";

if(msg == WM\_ASKCBFORMATNAME) return "WM\_ASKCBFORMATNAME";

if(msg == WM\_CHANGECBCHAIN) return "WM\_CHANGECBCHAIN";

if(msg == WM\_HSCROLLCLIPBOARD) return "WM\_HSCROLLCLIPBOARD";

if(msg == WM\_QUERYNEWPALETTE) return "WM\_QUERYNEWPALETTE";

if(msg == WM\_PALETTEISCHANGING) return "WM\_PALETTEISCHANGING";

if(msg == WM\_PALETTECHANGED) return "WM\_PALETTECHANGED";

if(msg == WM\_HOTKEY) return "WM\_HOTKEY";

if(msg == WM\_PRINT) return "WM\_PRINT";

if(msg == WM\_PRINTCLIENT) return "WM\_PRINTCLIENT";

if(msg == WM\_APPCOMMAND) return "WM\_APPCOMMAND";

if(msg == WM\_THEMECHANGED) return "WM\_THEMECHANGED";

if(msg == WM\_CLIPBOARDUPDATE) return "WM\_CLIPBOARDUPDATE";

if(msg == WM\_DWMCOMPOSITIONCHANGED) return "WM\_DWMCOMPOSITIONCHANGED";

if(msg == WM\_DWMNCRENDERINGCHANGED) return "WM\_DWMNCRENDERINGCHANGED";

if(msg == WM\_DWMCOLORIZATIONCOLORCHANGED) return "WM\_DWMCOLORIZATIONCOLORCHANGED";

if(msg == WM\_DWMWINDOWMAXIMIZEDCHANGE) return "WM\_DWMWINDOWMAXIMIZEDCHANGE";

if(msg == WM\_GETTITLEBARINFOEX) return "WM\_GETTITLEBARINFOEX";

return "";

}

void logMessage(string prefix, HWND hWnd, uint uMsg, WPARAM wParam, LPARAM lParam)

{

string msg = msg\_toString(uMsg);

if(msg.length == 0)

msg = tryformat("%x", uMsg);

logCall("%s(hwnd=%x, msg=%s, wp=%x, lp=%x)", prefix, hWnd, msg, wParam, lParam);

}

module workaround;

import std.conv;

///////////////////////////////////////////////////////////////

// fix bad compilation order, causing inner function to be generated

// before outer functions (bugzilla 2962)

// this must be \*parsed\* before other usage of the template

static if(\_\_traits(compiles,std.conv.parse!(real,string))){}

//\_\_gshared int[string] x;

version(none) debug

{

// drag in some debug symbols from libraries (see bugzilla ????)

extern extern(C)

{

\_\_gshared int D3vdc4util8TextSpan6\_\_initZ;

\_\_gshared int D3sdk4port4base4GUID6\_\_initZ;

\_\_gshared int D3std4json9JSONValue6\_\_initZ;

}

shared static this()

{

auto x1 = &D3vdc4util8TextSpan6\_\_initZ;

auto x2 = &D3sdk4port4base4GUID6\_\_initZ;

auto x3 = &D3std4json9JSONValue6\_\_initZ;

}

}

// This file is part of Visual D

//

// Visual D integrates the D programming language into Visual Studio

// Copyright (c) 2010 by Rainer Schuetze, All Rights Reserved

//

// Distributed under the Boost Software License, Version 1.0.

// See accompanying file LICENSE\_1\_0.txt or copy at <http://www.boost.org/LICENSE_1_0.txt>

module visuald.xmlwrap;

version(D\_Version2)

{

private static import std.xml;

alias std.xml.Element Element;

alias std.xml.Document Document;

alias std.xml.XMLException XmlException;

alias std.xml.CheckException RecodeException;

Element[] elementsById(Element elem, string id)

{

Element[] elems;

foreach(e; elem.elements)

if(e.tag && e.tag.name == id)

elems ~= e;

return elems;

}

string getAttribute(Element elem, string attr)

{

if(string\* s = attr in elem.tag.attr)

return \*s;

return null;

}

void setAttribute(Element elem, string attr, string val)

{

elem.tag.attr[attr] = val;

}

Element getRoot(Document doc)

{

return doc;

}

Element getElement(Element e, string s)

{

foreach(el; e.elements)

if(el.tag.name == s)

return el;

return null;

}

Document newDocument(string root)

{

return new Document(new std.xml.Tag(root));

}

Document readDocument(string text)

{

//        setHWBreakpopints();

return new Document(text);

}

string[] writeDocument(Document doc)

{

return doc.pretty(1);

}

alias std.xml.encode encode;

}

else

{

private static import xmlp.pieceparser;

private static import xmlp.xmldom;

private static import xmlp.input;

private static import xmlp.delegater;

private static import xmlp.format;

private static import xmlp.except;

private static import inrange.recode;

alias xmlp.xmldom.Element Element;

alias xmlp.xmldom.Document Document;

alias xmlp.except.XmlException XmlException;

alias inrange.recode.RecodeException RecodeException;

Element[] elementsById(Element elem, string id)

{

return elem.elementById(id);

}

string getAttribute(Element elem, string attr)

{

return elem[attr];

}

void setAttribute(Element elem, string attr, string val)

{

elem[attr] = val;

}

Element getRoot(Document doc)

{

return doc.root;

}

Element getElement(Element e, string s)

{

int idx = e.firstIndexOf(s);

if(idx >= 0)

if(Element el = cast(Element) e.children[idx])

return el;

return null;

}

Document newDocument(string root)

{

return new Document(new Element(root));

}

Document readDocument(string text)

{

auto spi = new xmlp.pieceparser.XmlParserInput(inrange.instring.dcharInputRange(text));

Document doc = xmlp.pieceparser.XmlPieceParser.ReadDocument(spi);

return doc;

}

string[] writeDocument(Document doc)

{

xmlp.format.XmlFormat canit = new xmlp.format.XmlFormat();

canit.indentAdjust = 1;

// Pretty-print it

string[] result;

canit.canonput(doc, result);

return result;

}

}