Sequence of calls involved in one log entry.
Manually worked out by reading the code.
Flash => sim.exe => CNCPlotterDialog.dll.

immersive_log MILL PLOTTER.txt

```
13 <43436> = >Flash FS Command Call> - <invoke name="editor_group"
returntype="xml"><arguments><string>set_active_program setActiveProgramCallback O30007</string></
arguments></invoke>13 <43436> = >Flash FS Command Call> - <invoke name="editor_group"
returntype="xml"><arguments><string>set_active_program setActiveProgramCallback O30007</string></
arguments></invoke>
```

Flash

biz\immersive\ControllerModule\HaasController\LatheV16\screens\ProgramEditScreen.as

```
function DisplaySelectedProgram( ):void
{
	var ret:String = "";
	this.SetDefaultMode();
	this._program_list_View.SelectCurrentHighlitedProgram();
	ret = this._program_list_View.GetCurrentHighlitedProgram();
	this.NCcontroller.operator_panel.program_list.SetCurrentProgram ( ret );
	this.MakeProgramListInvisible();
	this.SetFonts();
	this.UpdateProgramDisplay();
	this.SetCurrentProgram();
}
```

biz\immersive\ControllerModule\OperatorModule\ProgModule\ProgramList.as

```
public function SetCurrentProgram( progr_name:String ):void
{
	this.SetCurrentControllerProgram( progr_name );
	_ctrl.sim_com.setActiveProgramCall( this.current_program_name );
}
```

biz\immersive\ControllerModule\EICallbackClass.as

```
public function setActiveProgramCall( programName:String ):void
{
	if( local_mode )
	{
		this.setActiveProgramCallback( programName, "0" );
		return;
	}

	var callConstruct:String = "set_active_program setActiveProgramCallback " + programName;
	this.MakeExternalCall(	"editor_group", callConstruct);
}
```

C++ (sim.exe)

SIM_MAIN/flash_syntax.h

```
#define SET_ACTIVE_PROGRAM_EDITOR_GROUP "set_active_program"
```

SIM_MAIN\FlexVCBridgeDlg.cpp

```
// Receive events from Flash:

BEGIN_EVENTSINK_MAP(FlexVCBridgeDlg, CDialog)
        ON_EVENT(FlexVCBridgeDlg, IDC_SHOCKWAVEFLASH_NEW, 197,
FlexVCBridgeDlg::FlashCallShockwaveflash1, VTS_BSTR)
END_EVENTSINK_MAP()

void FlexVCBridgeDlg::FlashCallShockwaveflash1(LPCTSTR request)
{
        ProcessFlashCall( request );
}

void FlexVCBridgeDlg::ProcessFlashCall(LPCTSTR request)
{
        CMarkup xml;
        // Non blocking call
        aw = new ExternalFlashThreadCom( action_string, arg_string );
}
```

SIM_MAIN\ExternalFlashThreadCom.cpp

```
void ExternalFlashThreadCom::execute(void)
{
        process_ie_call( this->ActionString, this->ArgString );
}

void process_ie_call( CString AString, CString Arg )
{
        if( AString.CompareNoCase( EDITOR_GROUP ) == 0 )
        {
                if( Arg.GetLength() != 0 )
                        ie_process_editor_group( Arg , true );
                return;
        }
```

SIM_MAIN\ie_call_editor_group.cpp

```
void ie_process_editor_group(CString string_to_parse, int callback_flag)
{
   if (temp_str.CompareNoCase(_T(SET_ACTIVE_PROGRAM_EDITOR_GROUP)) == 0) { // process flash
requests to load a file on the local hard drive
       ie_editor_group_set_active_prgm(next_args, callback_flag);
   }
}

void ie_editor_group_set_active_prgm( CString string_to_parse , int callback_flag)
{
        prg_name = string_to_parse.Tokenize(" ",curPos);
        global_p_MainController->set_current_nc_program( prg_name );
}
```

SIM_MAIN\easyrob_controller.cpp

```
void easyrob_controller::set_current_nc_program( CString prog_name )
{
        this->current_nc_program = this->TheProgramList->set_current_program( prog_name );
        if( this->current_nc_program == NULL )
```

```
                return;
        this->current_nc_program->update_plotter();
        return;
}
```

SIM_MAIN\program_list.cpp

```
program_storage *program_list::set_current_program( CString new_program )
{
        program_storage *aprogram;
        char prg_name[256];
        strcpy( prg_name , (LPCTSTR) new_program );
        aprogram = program_list::find_program( prg_name );
        if( aprogram == NULL )
                return aprogram;
        program_list::current_active_prg = aprogram;
        return aprogram;
};
```

SIM_MAIN\program_storage.cpp

```
int program_storage::update_plotter ( void )
{
        int ret = 0;
        global_p_MainController->m_jobQ->addJob( new PreviewNCProgram( this ) );
        return ret;
}

PreviewNCProgram::PreviewNCProgram( program_storage *a_prg )
{
        this->the_prg = a_prg;
}

void PreviewNCProgram::execute(void)
{
        if( this->the_prg == NULL )
                return;

        int ret  = this->the_prg->update_plotter_thread();
}

int program_storage::update_plotter_thread(void)
{
   int ret = 0;
   if (global_p_MainController == NULL)
      return ret;
   CArray<const char*> c;
   CritSectEx::Scope scope; // empty constructor doesn't lock anything
   scope.Lock(program_cs);
   this->UpdateProgramPositions();

   //  Program                  Program preview
   //  Modified                 Created
   // {_program_modified)       (dirty_flag)            What to do
   //
   //    0                            TRUE                    Create Program Preview Send Program
   //    1                            TRUE                    Create Program Preview Send Program
   //    0                            FALSE                   Just send Program
   //    1                            FALSE                           Create Program Preview Send Program

   if (this->_program_modified == 1 || this->preview_path.GetNumBlocks() == 0) { // Create Preview
      this->preview_path.ClearList();
      ret = global_p_MainController->create_program_preview_path(this);
```

```
        this->preview_path.dirty_flag = true;
        this->_program_modified = 0;
        ret = this->SendPreviewPathTOPlotter(); // Send Program
        this->preview_path.dirty_flag = false;
    }
    else { // just set new program
        if (this->preview_path.GetNumBlocks() != 0) {
            CncPlotterDialog_SetProgramPath(global_p_MainController->machine_name.GetBuffer(),
( immersive::CncProgramPath* ) &this->preview_path);
        }
    }

    if (global_p_MainController->PreviewCompletedCallback.GetLength() != 0) {
        c.Add(ret ? "1" : "0");
        API_CallBlockingFlashFunction(global_p_MainController->PreviewCompletedCallback.GetBuffer(), c);
    }

    scope.Unlock(); // We are done Unlock
    return ret;
}

int program_storage::SendPreviewPathTOPlotter ( void )
{
            global_p_MainController->motion_engine.UpdatePreviewWorkOffsets();
            global_p_MainController->motion_engine.UpdatePreviewToolOffsets();
            if( this->preview_path.GetNumBlocks() != 0 )
            {
                        CncPlotterDialog_SetProgramPathWorkToolOffset(        global_p_MainController-
>machine_name.GetBuffer() ,
                                                                               (immersive::CncProgramPath *)&this-
>preview_path ,
                                                                               &(global_p_MainController-
>motion_engine.preview_workoffset_list) ,
                                                                               &(global_p_MainController-
>motion_engine.preview_tooloffset_list) );
            }
}
```

C++ (CNCPlotterDialog.dll)

CNCPlotterDialog\CncPlotterDialog.cpp

```
CNC_PLOTTER_DIALOG_API void CncPlotterDialog_SetProgramPathWorkToolOffset( const std::string&
machine, CncProgramPath* path,  WorkOffsetList* work_offsets,ToolGeometryFileList* tools)
{
            AFX_MANAGE_STATE( AfxGetStaticModuleState() );
            CCncPlotterDialogDlg* dialog = theApp.GetCncPlotterDialogWindow();

            if ( dialog )
                        dialog->SetProgramPathWorkToolOffset( machine, path, work_offsets, tools );
}

void CCncPlotterDialogDlg::SetProgramPathWorkToolOffset( const std::string& machine,
immersive::CncProgramPath* path,  immersive::WorkOffsetList* work_offsets, immersive::ToolGeometryFileList*
tools )
{
            if ( this->mOSG )
                        this->mOSG->SetProgramPathWorkToolOffset( machine, path, work_offsets, tools );
}


CNC_PLOTTER_DIALOG_API void CncPlotterDialog_SetProgramPath( const std::string& machine,
CncProgramPath* path )
{
	AFX_MANAGE_STATE( AfxGetStaticModuleState() );
	CCncPlotterDialogDlg* dialog = theApp.GetCncPlotterDialogWindow();

	if ( dialog )
	{
		dialog->SetProgramPath( machine, path );
	}
}
```

CNCPlotterDialog\CncPlotterDialogDlg.h

```
   void SetProgramPath( const std::string& machine, immersive::CncProgramPath* path ) { if ( this->mOSG ) this->mOSG->SetProgramPath( machine, path ); }
```

CNCPlotterDialog\MFC_OSG.h

```
typedef enum {
	ZERO_EVENT = 0,
	STOP_PROGRAM,
	START_PROGRAM,
	START_PROGRAM_NAME,
	UPDATE_FINAL_PART,
	UPDATE_TOOL_LIST,
	UPDATE_WORKOFFSET_LIST,
	UPDATE_PATH_WORK_TOOL_LIST,
	UPDATE_CNC_PATH,
	HIDE_FINAL_PART,
	RESET_EVENT,
	WINDOW_SIZING_EVENT,
	CLEAR_SLIDE_BAR_TIMER_EVENT,
	CLEAR_ACTIVE_PROGRAM_EVENT,
	MAX_NUMBER_OF_EVENTS
} AnimationEventType;
```

CNCPlotterDialog\MFC_OSG.cpp

```
void cOSG::SetProgramPathWorkToolOffset( const std::string& machine, immersive::CncProgramPath* path, immersive::WorkOffsetList* work_offsets, immersive::ToolGeometryFileList* tools )
{
   this->_current_machine = machine;
   this->_current_path = path;
   this->_work_offsets = work_offsets;
   this->_tools = tools;

   EventItem a_new_event;
   a_new_event.SetEventType( UPDATE_PATH_WORK_TOOL_LIST );

   this->_event_mgr.PushEvent( a_new_event );
}

void cOSG::SetProgramPath( const std::string& machine, immersive::CncProgramPath* path )
{
   this->_current_machine = machine;
   this->_current_path = path;

   EventItem a_new_event;

   a_new_event.SetEventType( UPDATE_CNC_PATH );
```

```
    this->_event_mgr.PushEvent( a_new_event );
}


void cOSG::PreFrameUpdate()
{
    EventItem a_event;

           int t1 = 0;
           int t2 = 0;
           int t3 = 0;
           int t4 = 0;

    // Due any preframe updates in this routine
    while( _event_mgr.AnyEventsToProcess() )
    {
        a_event = this->_event_mgr.PopEvent();

       switch( a_event.GetEventType () )
       {
           case UPDATE_PATH_WORK_TOOL_LIST:

               this->_scene.SetWorkOffsets( this->_machine, this->_work_offsets );

               this->_scene.SetToolGeometryFileList( this->_machine, this->_tools );

               if( this->_current_path != NULL )
               {
                   this->_scene.ProgramReset();

                   this->_scene.SetProgramPath( this->_current_machine, this->_current_path );
                   this->_scene.SetCurrentProgramFinalPartDisplay( this->_enable_final_flag );

                   this->ResetActiveProgramView();
               }
           break;
           case UPDATE_CNC_PATH:
               if( this->_current_path != NULL )
               {
                   this->_scene.ProgramReset();

                   this->_scene.SetProgramPath( this->_current_machine, this->_current_path );
                   this->_scene.SetCurrentProgramFinalPartDisplay( this->_enable_final_flag );

                   this->ResetActiveProgramView();
               }

           break;
}
```

CNCPlotterDialog\SceneManager.cpp

```
void SceneManager::SetProgramPath( const std::string& machine, immersive::CncProgramPath* path )
{
           this->_previewPath = *path;
           this->grepProgramPath();
}

void SceneManager::grepProgramPath( void )
{
           this->_progMgr.AddProgram( this->_previewPath );
           this->ProgramReset();

}
```

```cpp
void SceneManager::ProgramReset( void )
{
    osg::Vec3 pos( 0.0f, 0.0f, 0.0f );
    osg::Vec3 workOffset( 0.0f, 0.0f, 0.0f );
    int toolId( 0 );
    this->ProgramStop();
    this->_progMgr.ResetActiveProgramStepCounter( pos, workOffset, toolId );
    this->_toolMgr.MoveTool( pos );
    this->SetUpdate();

    this->_tc->MoveToTime( 0.0f );
}
```