TYX Corporation

Productivity Enhancement Systems

Reference	TYX_0051_11
Revision	1.0
Document	ConfControl.doc
Date	August 08, 2004

Configuration Control Management

1 Project configuration:

1.1 Introduction:

In a complete test station, you will have at least one of the following:

1. A **Paws project** with the **.paw** extension.

2. An **Atlas TPS**. This file has the default **.atl** suffix, although it is possible to have another suffix or no suffix at all (in order to support legacy files). You may have one or more Atlas file, with one main program and one or more modules.

3. A **Device database file**. This file contains the description of your station with the exception of the switching and the interface adapter. This file is typically unique to the station and remains with all projects on that station. You can have one file, or you can have several files in one project, each one describing each individual instrument in the station. This, or those files have the default **.ddb** suffix, although it is possible to have another suffix or no suffix at all (in order to support legacy files).

This, or these files may also contain TYX Macro code drivers in the event that you chose to implement the drivers using TYX Macro code as opposed to C/C++ encoded drivers.

4. A **Switch database file**. This file contains the description of the switches and the wires that stay with the station. This file typically is unique to the station and remains with all projects on that station. This file has the default **.sdb** suffix, although it is possible to have another suffix or no suffix at all (in order to support legacy files).

5. An **Ita database file**. This file contains the wiring in the Interface Test Adapter. This file is typically associated with the TPS files. This file has the default **.ita** suffix although it is possible to have another suffix or no suffix at all (in order to support legacy files).

In the event that you have a C/C++ driver or drivers, you will also have the following files:

6. **CEM File(s)**. This driver file (**.dll**) will be built based on several source files with the **.c/.cpp**, **.h** and possibly the **.rc** extension. These driver files are typically unique to a station. These files are needed in the event that there are

no TYX Macro code drivers, although they may coexist i.e., you can have instruments with TYX Macro code driver, some with CEM drivers and even some MATE devices that do not require for the user to write any driver, all in the same project. With the paws environment, you will have some additional files:

a. A file with the **.dfw** extension. This file will contain the Wcem Wizard information including the function mapping.

b. If you care to have a version for the dll that will be generated after building the CEM files, you will also have a file with the **.rc** extension.

c. If you generated an MSVC++6.0 project to build the CEM dll, you will have a couple of files with the extensions: **.dsw** and **.dsp**. Those files will allow you to debug the CEM driver at runtime. For more information, please refer to document **TYX_0051_4**. If you used the .NET Studio, you will have files with the **.sln** and **.vcproj** extensions.

Note: Some files generated by the CEM wizard should not be edited:

- **Wrapper.c** or **Wrapper.cpp**.
- **Key.h**

Some files are intended to be edited, but only for the content of the functions themselves. The user should not manually edit the name of the function or the comments outside of the function definition. This is true for the files generated by the wizard:

- **<device name>.c** or **<device name>.cpp**.
- **error.c** or **error.cpp**.
- **ctlr.c** or **ctlr.cpp**.

In the example below that came out of ctlr.cpp, the text in blue/bold should not be manual edited.

```
//BEGIN{DFW}:CONNECT

int CCALLBACK doLoad ()

//END{DFW}

{
        // user-stub function call

        userStubNaaCONNECT();


        // please insert your CEM driver code here


        return 0;

}
```

Everything within the curly brackets is intended to include the user code.

Once you have built the project from Paws Studio, you will have a few binaries that may be of interest to you. The building process is described in full detail in the online help and there is a very useful diagram in the following location. **TPS User's Guide, PART 1, Section 1, TEST SET MODELING, Section 1, Test Set Modeling Overview, Page 6**:

7. Two files with your project name with the **.DAT** and the **.OBJ** extension. You may have a **PAX** file as well, depending upon the options that you used for the building process.

8. You have miscellaneous files that are used for optional tools such as the RTDG or in the process of debugging the Atlas code or the TYX Macro code:

- **SYM** file.
- **SIG** file.
- **SWX** files.
- **DeviceDB.DEV** file.
- **ATL** source file(s)
- **DDB** source file(s)
- **KWD** files that are not part of the building process, but part of the environment. They can be found **in <usr>\tyx\tab**.
- **LexDB.lex** file

Once you have built and run the TPS, you will have a few files that may present some interest to you:

9. A file with the **.ehf** extension. This file will contain a history of what you entered manually at runtime. This may allow you to avoid having to re-enter manually multiple values that you have already entered.

10. A **Log** file. The content of this file can be configured in the options for the Wrts. The name and the location of it may also be configured. The default name is **Log** or **Log.**txt depending upon the Paws Studio version, and the default location for the log file is in the folder with the paw project.

11. A **PRINTER** file. The content of which would contain lines that have been entered by the TPS and have been chose to be sent to a file, which is what happens by default.

Important Note: If you make changes to the lex files, you will need to rebuild the entire project. This is made clear in the TPS User's Guide build diagram referred to above since the **LexDB.LEX** is a input to the building process for:

- The Atlas file(s)
- The device database file(s)
- The switch database file
- The ita file and
- The CEM driver environment.

You will first need to reopen the **CEM wizard** so that the **key.h** and the **wrapper.c(pp)** can be regenerated and then you should do a rebuild all. You should follow this procedure with your own procedure to distribute potential binary files to various other projects that use binary files as an input as opposed to source files.

1.2 The Station files:

1.2.1 Locations:

The files below typically stay with the station regardless of the TPS, the ITA or the UUT that the station is testing. Here is a list of files that are associated to the station:

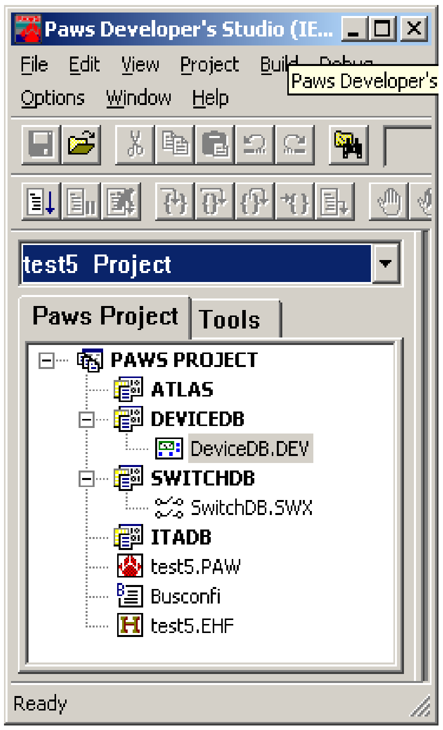
1.2.1.1 The Device Database(s):

The source files usually ends with the **.ddb** extension. This is the highly recommended extension for this type of file. The Paws Studio can accept other extensions and even no extension in order to support legacy files. You can have one or more files. This file, or these files, will contain:

- The static description that will describe the devices in the station except for the resources associated to the switches.
- The TYX Macro code drivers. If you have implemented your drivers using the TYX Macro code, you do not need an external C/C++ compiler and you do not need to generate any dlls, which is required when you use C/C++ drivers.
- The binary file that results from building the Device Database file(s) is **DeviceDB.dev**. Those Device Database source files can be built into a **DeviceDB.dev** file from a project that does not include any other types of source files (no Atlas files, no Switch Database file and no Ita file).
- Building the .dev separately can be useful when you want to generate one binary file and have all the other project include this **.dev** binary as an input to their building process. That will prevent other projects from making any undesirable changes to the ddb source code.
- When including a .dev in a project, you have two options:

  i. Include the .dev file where it is located

  ii. Include the default .dev that has to be located in the **<usr>\tyx\sub\<Subset>\<Station>\station** folder. This is the case in the following picture:



You can determine the location of the files by right clicking on them and checking on the settings.

1.2.1.2 The Switch Database:

This file has the **.sdb** extension. This is the highly recommended extension for this type of file. The Paws Studio can accept other extensions and even no extension in order to support legacy files.

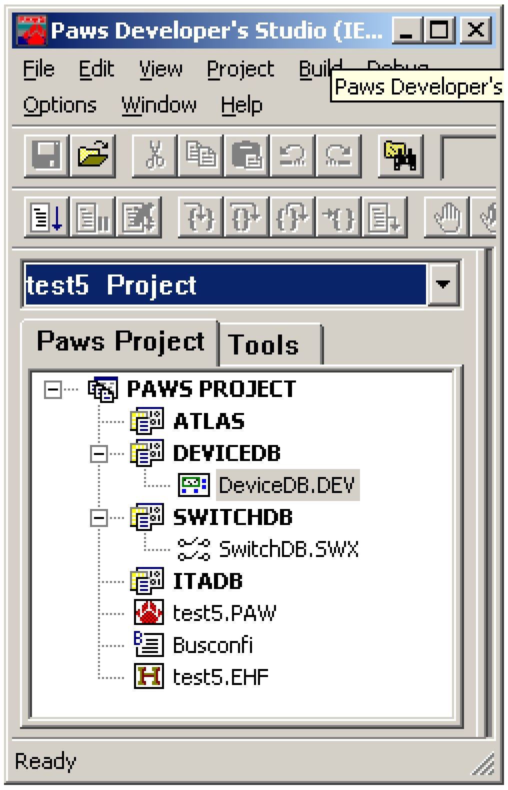
The binary file that results from building the Switch Database file is the SwitchDB.swx file. Just like for the device database file(s), you can also build the SwitchDB.swx file from the Switch Database source file in a project without including any other types of files in your project (no Atlas file, no Device Database file and not Ita file).

Similarly to the database file, you can chose to generate the SwitchDB.swx from one project and have all other projects include this .swx binary as an input to their building process. That will prevent them from making any undesirable changes to the sdb source code. This is sometimes done in the same project that builds the .dev file for distribution, often referred to as a master project.

When including a .dev in a project, you have two options:

iii. Include the .dev file where it is located

iv. Include the default .dev that has to be located in the **<usr>\tyx\sub\<Subset>\<Station>\switch** folder. This is the case in the following picture:



1.2.1.3 The C\C++ driver files:

Those files have the **.h**, **.c** or **.cpp** extension, and possibly a **.rc** file if you chose to include one. You have the ability to build a CEM dll or multiple CEM dlls. The default name for the CEM dll is **Wcem.dll**. You may however give your dll(s) the name(s) that you want.

Note: It is not recommended to use the name **CEM** for the name given to the CEM module or to be building a CEM dll called **cem.dll**. This will lead to building problems unless you are an advanced user and really know what you are doing.

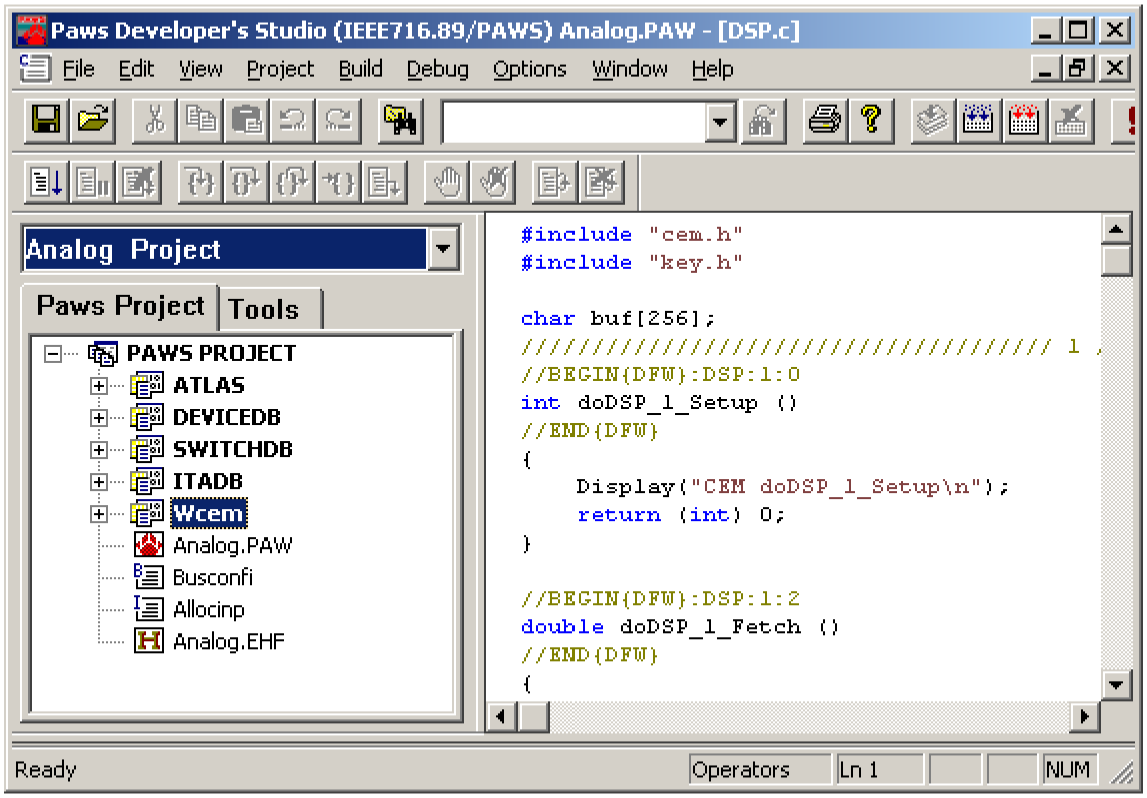
1.2.1.3.1 One CEM dll:

Why would you want one CEM dll?

It may be a contractual requirement.

It may also be considered that in your environment it is easier to manage and move around: You only have one CEM dll file to worry about.

In this case, you only need one CEM module in your project as illustrated in the following image:



```
#include "cem.h"
#include "key.h"

char buf[256];
////////////////////////////////////////// 1 ,
//BEGIN{DFW}:DSP:1:0
int doDSP_1_Setup ()
//END{DFW}
{
    Display("CEM doDSP_1_Setup\n");
    return (int) 0;
}

//BEGIN{DFW}:DSP:1:2
double doDSP_1_Fetch ()
//END{DFW}
{
```

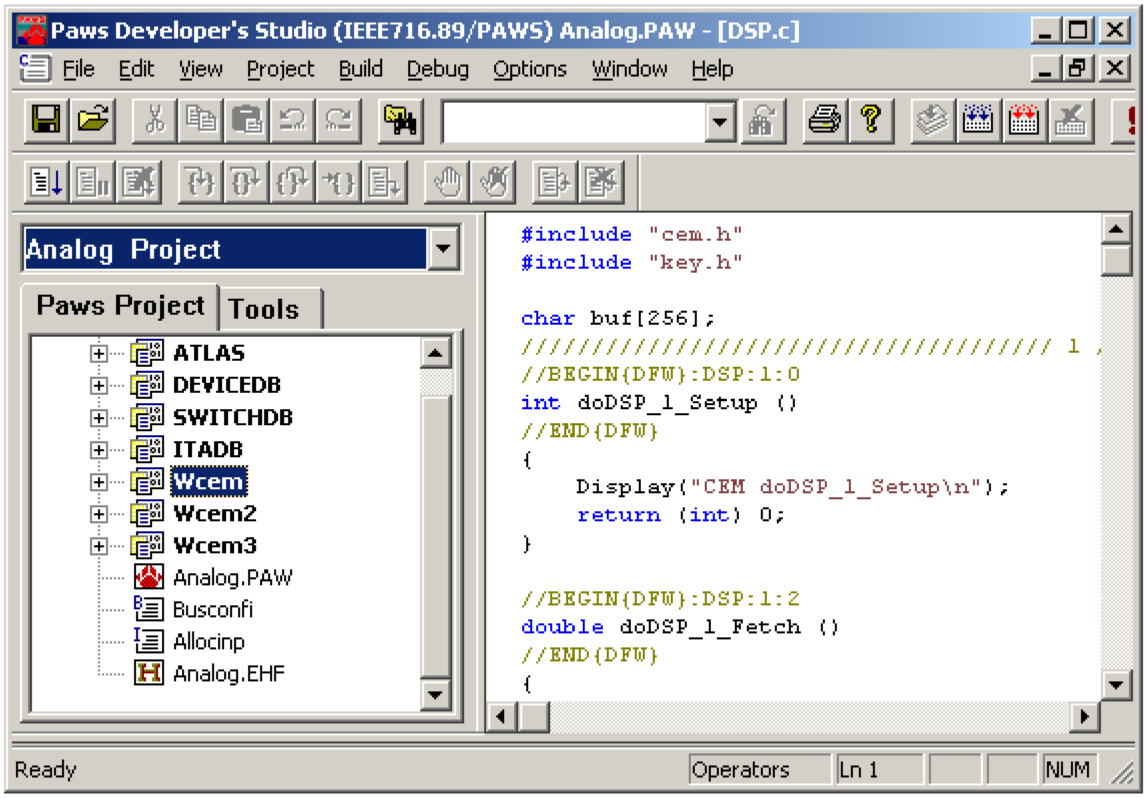
1.2.1.3.2 Multiple CEM dll:

Why would you want to have multiple CEM dlls?

- It may be a contractual requirement.
- It may also be because it is consider to be easier to maintain: You can easily have one dll per device. Maintaining and updating the dll for one device does not involve rebuilding a dll that includes all the files for all the other devices, which may not even be ready to build. This allows different people to build different dlls independently from what anyone else is doing. The downside to this is that you will have a lot more dlls to manage and to keep track of.

Note: It is also possible to have a CEM dll support multiple devices but not all the devices. You would still have multiple dlls, but not as many. It could be for instance deemed useful to have one dll per programmer.

You may then have one project with multiple CEM modules, or multiple projects with just one CEM module. Paws Developer Studio will generate one CEM dll per CEM module. The following image has 3 CEM modules:



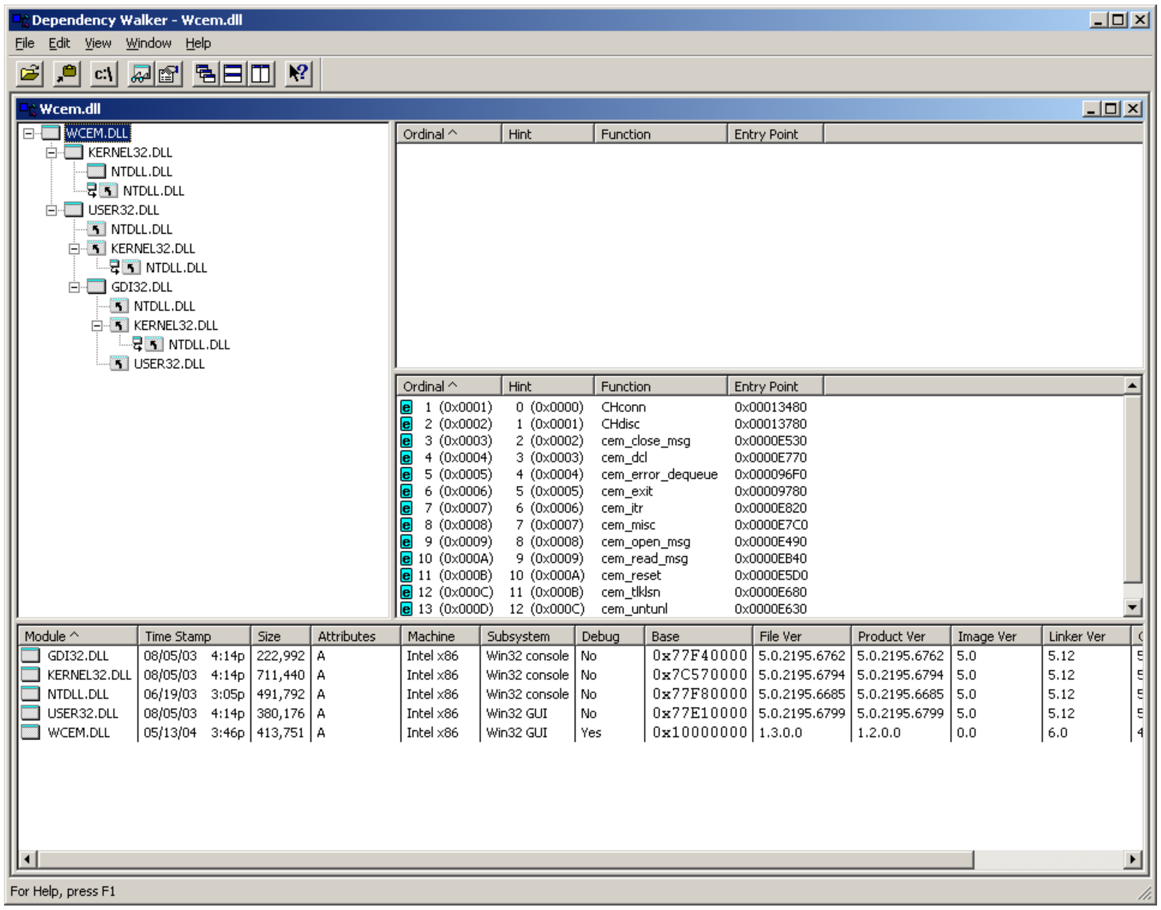
```
#include "cem.h"
#include "key.h"

char buf[256];
/////////////////////////////////////// 1 /
//BEGIN{DFW}:DSP:1:0
int doDSP_1_Setup ()
//END{DFW}
{
    Display("CEM doDSP_1_Setup\n");
    return (int) 0;
}

//BEGIN{DFW}:DSP:1:2
double doDSP_1_Fetch ()
//END{DFW}
{
```

1.2.1.3.3 Good to know in general:

- **<u>Generating the CEM dll:</u>** In the CEM settings, you can set the location where the CEM dll will be generated, using either an absolute path, or a path relative to the project location.
- **<u>Third party dlls:</u>** For each CEM dll, you may have more than one dll to worry about in the event that, for instance, you connect to a Pnp driver. Then, outside of the TYX environment, you also have to worry about the dlls that your code in the CEM dll is connecting to. For more information on those, you may want to refer to the instructions that come with the Pnp driver(s).
- **<u>Dependency check:</u>** In order to see what dll your CEM dll is linked to statically, you can use the Microsoft tool that comes with the compiler called *Depends*. Please note that you will not be able to see what dlls are dynamically linked to the CEM dll until you try to run your project.
    - Here is an example of that tool that opened a CEM dll called WCEM.dll and wasn't connecting to any Pnp drivers. This **WCEM.**dll appears to not have any problems with static linking. Note how it still uses some environmental dlls such as User32.dll and more:



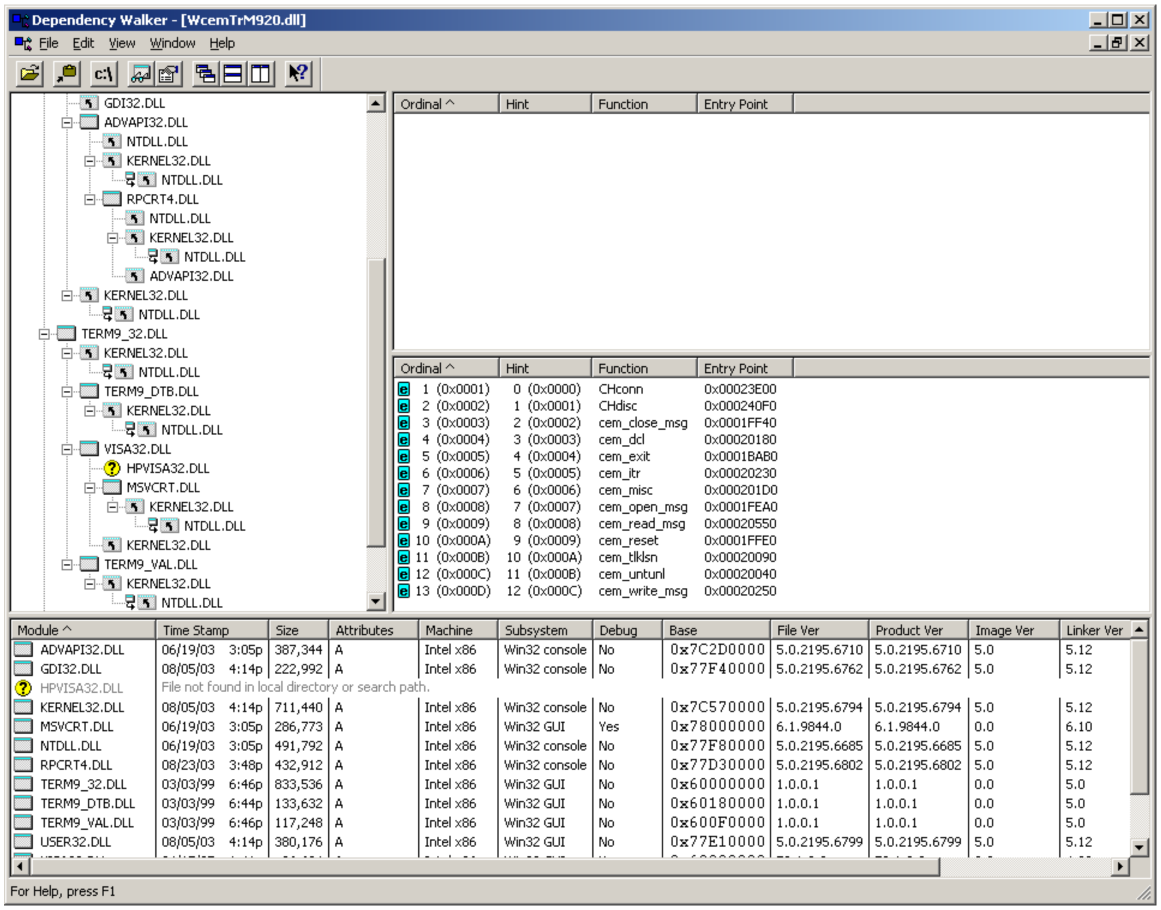
```
WCEM.DLL
    KERNEL32.DLL
        NTDLL.DLL
        NTDLL.DLL
    USER32.DLL
        NTDLL.DLL
        KERNEL32.DLL
            NTDLL.DLL
        GDI32.DLL
            NTDLL.DLL
            KERNEL32.DLL
                NTDLL.DLL
            USER32.DLL
```

Ordinal ^	Hint	Function	Entry Point

Ordinal ^	Hint	Function	Entry Point
1 (0x0001)	0 (0x0000)	CHconn	0x00013480
2 (0x0002)	1 (0x0001)	CHdisc	0x00013780
3 (0x0003)	2 (0x0002)	cem_close_msg	0x0000E530
4 (0x0004)	3 (0x0003)	cem_dcl	0x0000E770
5 (0x0005)	4 (0x0004)	cem_error_dequeue	0x000096F0
6 (0x0006)	5 (0x0005)	cem_exit	0x00009780
7 (0x0007)	6 (0x0006)	cem_itr	0x0000E820
8 (0x0008)	7 (0x0007)	cem_misc	0x0000E7C0
9 (0x0009)	8 (0x0008)	cem_open_msg	0x0000E490
10 (0x000A)	9 (0x0009)	cem_read_msg	0x0000EB40
11 (0x000B)	10 (0x000A)	cem_reset	0x0000E5D0
12 (0x000C)	11 (0x000B)	cem_tlklsn	0x0000E680
13 (0x000D)	12 (0x000C)	cem_untunl	0x0000E630

Module ^	Time Stamp	Size	Attributes	Machine	Subsystem	Debug	Base	File Ver	Product Ver	Image Ver	Linker Ver
GDI32.DLL	08/05/03 4:14p	222,992	A	Intel x86	Win32 console	No	0x77F40000	5.0.2195.6762	5.0.2195.6762	5.0	5.12
KERNEL32.DLL	08/05/03 4:14p	711,440	A	Intel x86	Win32 console	No	0x7C570000	5.0.2195.6794	5.0.2195.6794	5.0	5.12
NTDLL.DLL	06/19/03 3:05p	491,792	A	Intel x86	Win32 console	No	0x77F80000	5.0.2195.6685	5.0.2195.6685	5.0	5.12
USER32.DLL	08/05/03 4:14p	380,176	A	Intel x86	Win32 GUI	No	0x77E10000	5.0.2195.6799	5.0.2195.6799	5.0	5.12
WCEM.DLL	05/13/04 3:46p	413,751	A	Intel x86	Win32 GUI	Yes	0x10000000	1.3.0.0	1.2.0.0	0.0	6.0

- This following example, is an example where a dll cannot be found


Dependency Walker - [WcemTrM920.dll]
File Edit View Window Help
For Help, press F1

```
      GDI32.DLL
    ADVAPI32.DLL
      NTDLL.DLL
      KERNEL32.DLL
        NTDLL.DLL
      RPCRT4.DLL
        NTDLL.DLL
        KERNEL32.DLL
          NTDLL.DLL
        ADVAPI32.DLL
    KERNEL32.DLL
      NTDLL.DLL
  TERM9_32.DLL
    KERNEL32.DLL
      NTDLL.DLL
    TERM9_DTB.DLL
      KERNEL32.DLL
        NTDLL.DLL
    VISA32.DLL
      HPVISA32.DLL
      MSVCRT.DLL
        KERNEL32.DLL
          NTDLL.DLL
      KERNEL32.DLL
    TERM9_VAL.DLL
      KERNEL32.DLL
        NTDLL.DLL
```

Ordinal ^	Hint	Function	Entry Point

Ordinal ^	Hint	Function	Entry Point
1 (0x0001)	0 (0x0000)	CHconn	0x00023E00
2 (0x0002)	1 (0x0001)	CHdisc	0x000240F0
3 (0x0003)	2 (0x0002)	cem_close_msg	0x0001FF40
4 (0x0004)	3 (0x0003)	cem_dcl	0x00020180
5 (0x0005)	4 (0x0004)	cem_exit	0x0001BAB0
6 (0x0006)	5 (0x0005)	cem_itr	0x00020230
7 (0x0007)	6 (0x0006)	cem_misc	0x000201D0
8 (0x0008)	7 (0x0007)	cem_open_msg	0x0001FEA0
9 (0x0009)	8 (0x0008)	cem_read_msg	0x00020550
10 (0x000A)	9 (0x0009)	cem_reset	0x0001FFE0
11 (0x000B)	10 (0x000A)	cem_tlklsn	0x00020090
12 (0x000C)	11 (0x000B)	cem_untunl	0x00020040
13 (0x000D)	12 (0x000C)	cem_write_msg	0x00020250

Module ^	Time Stamp	Size	Attributes	Machine	Subsystem	Debug	Base	File Ver	Product Ver	Image Ver	Linker Ver
ADVAPI32.DLL	06/19/03 3:05p	387,344	A	Intel x86	Win32 console	No	0x7C2D0000	5.0.2195.6710	5.0.2195.6710	5.0	5.12
GDI32.DLL	08/05/03 4:14p	222,992	A	Intel x86	Win32 console	No	0x77F40000	5.0.2195.6762	5.0.2195.6762	5.0	5.12
HPVISA32.DLL	File not found in local directory or search path.										
KERNEL32.DLL	08/05/03 4:14p	711,440	A	Intel x86	Win32 console	No	0x7C570000	5.0.2195.6794	5.0.2195.6794	5.0	5.12
MSVCRT.DLL	06/19/03 3:05p	286,773	A	Intel x86	Win32 GUI	Yes	0x78000000	6.1.9844.0	6.1.9844.0	0.0	6.10
NTDLL.DLL	06/19/03 3:05p	491,792	A	Intel x86	Win32 console	No	0x77F80000	5.0.2195.6685	5.0.2195.6685	5.0	5.12
RPCRT4.DLL	08/23/03 3:48p	432,912	A	Intel x86	Win32 console	No	0x77D30000	5.0.2195.6802	5.0.2195.6802	5.0	5.12
TERM9_32.DLL	03/03/99 6:46p	833,536	A	Intel x86	Win32 GUI	No	0x60000000	1.0.0.1	1.0.0.1	0.0	5.0
TERM9_DTB.DLL	03/03/99 6:44p	133,632	A	Intel x86	Win32 GUI	No	0x60180000	1.0.0.1	1.0.0.1	0.0	5.0
TERM9_VAL.DLL	03/03/99 6:46p	117,248	A	Intel x86	Win32 GUI	No	0x600F0000	1.0.0.1	1.0.0.1	0.0	5.0
USER32.DLL	08/05/03 4:14p	380,176	A	Intel x86	Win32 GUI	No	0x77E10000	5.0.2195.6799	5.0.2195.6799	5.0	5.12

The name of the dll will be written up and a message will indicate that the file could not be found. In this case, **HPVISA32.dll** appears to be missing.

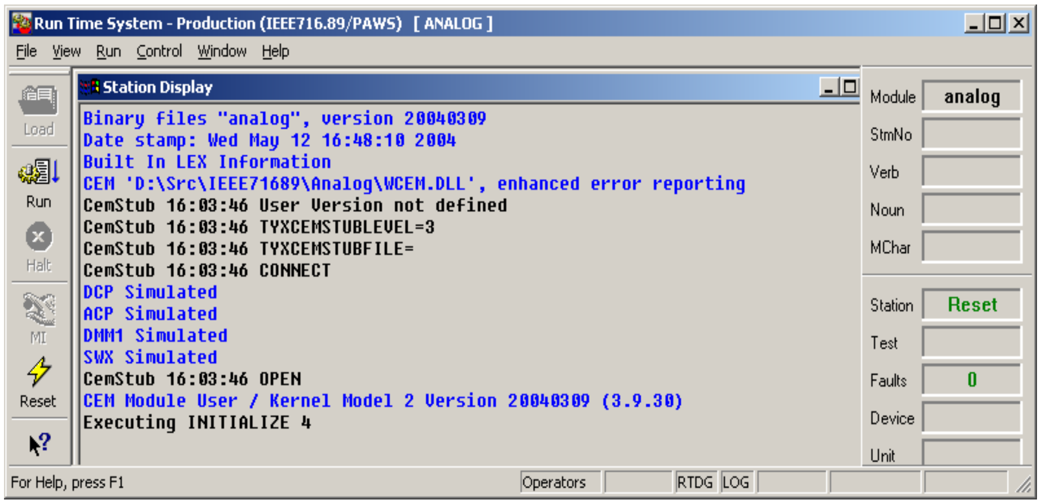
- o Even if the dll can be located, this tool will also report problems with functions within an existing dll. They can for instance be missing. That happens sometimes when the version of the dll we are connecting to is older than the version of the library we were linking to during the building process.

Please refer to the online help for that tool for further information.

- At runtime, if the CEM dll loads without any problems, you will have a message about that dll that will be something like:

CEM '<CEM path and name>', enhanced error reporting

as can be seen on the image below:

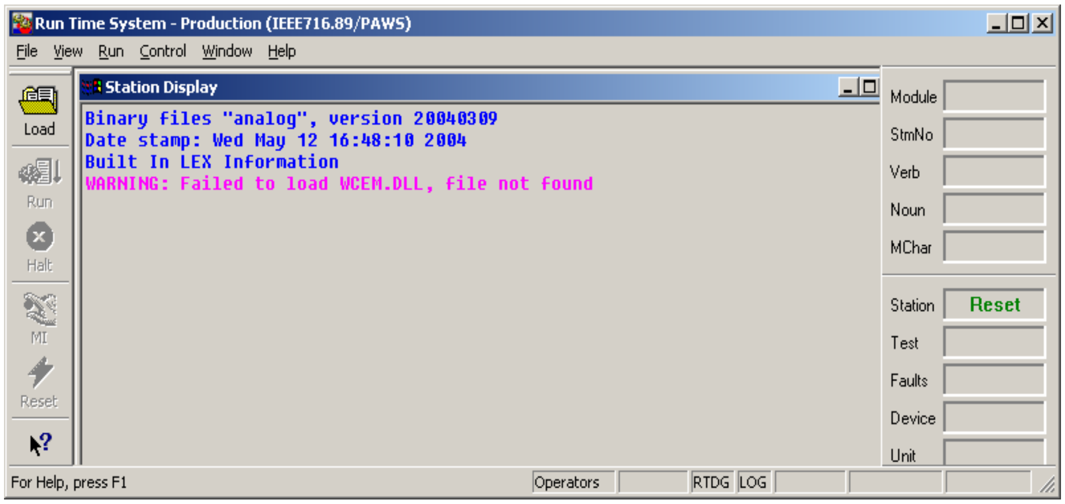


```
Binary files "analog", version 20040309
Date stamp: Wed May 12 16:48:10 2004
Built In LEX Information
CEM 'D:\Src\IEEE71689\Analog\WCEM.DLL', enhanced error reporting
CemStub 16:03:46 User Version not defined
CemStub 16:03:46 TYXCEMSTUBLEVEL=3
CemStub 16:03:46 TYXCEMSTUBFILE=
CemStub 16:03:46 CONNECT
DCP Simulated
ACP Simulated
DMM1 Simulated
SWX Simulated
CemStub 16:03:46 OPEN
CEM Module User / Kernel Model 2 Version 20040309 (3.9.30)
Executing INITIALIZE 4
```

If it can't find the CEM dll, you will get a warning such as

WARNING: Failed to load <CEM dll name>, file not found

as shown on the image below



```
Binary files "analog", version 20040309
Date stamp: Wed May 12 16:48:10 2004
Built In LEX Information
WARNING: Failed to load WCEM.DLL, file not found
```

If the file is found, but the dll can’t be loaded because it can’t find a subsequent dll, you will get a different message of the following format:

WARNING: Failed to load <path and name of CEM dll>, LoadLibrary API failed, The specified module could not be found.

1.3 The TPS files:

1.3.1 The Atlas file:

1.3.1.1 Location:

The Atlas file may be located anywhere you would like as far as the Paws Studio is concerned.

It is however common to have the atlas file in the following location:

<usr>\paws\<Your project folder>.

The legacy location would be to place the Atlas file in the following location: **<usr>\tyx\sub\<subset>\<station>\ATP**.

This legacy location may however not be convenient when you have multiple Atlas programs.

1.3.2 The Ita file:

This file is usually associated to an Atlas program and you can be expected to have multiple Ita files even if it is possible to reuse a particular Ita file for several Atlas files.

1.3.2.1 Location:

The Ita file may be located anywhere you would like as far as the Paws Studio is concerned.

It is however common to have the atlas file in the following location:

<usr>\paws\<Your project folder>\ ItaDB.

The default location for the Ita file with respect to the project folder is going to be **.\ItaDB**. The location of the Ita file can be changed when adding the Ita module to the .paw project.

It is also quite common to have the Ita file in the following location:

<usr>\paws\<Your project folder> or the same folder as your Atlas programs. You will then have to take care to not use the default folder when creating an Ita module in the Studio project.

The legacy location would be to place the Atlas file in the following location: **<usr>\tyx\sub\<subset>\<station>\ITA**. This legacy location may however not be convenient when you have multiple Ita files.

2 Managing the files for the Wrts:

2.1 Introduction:

As far as the binaries for the TPS, you have basically **3** options:

1. the **paw** project with the associated files,
2. the **.obj** file and associated files, and
3. the **.pax** project.

The first two will allow you to maintain different components of the project separately while the **pax** project will allow a more compact set of files making it harder to have the different components out of synchronization.

2.2 Paw project:

When loading the TPS in the Wrts, you should look for the **<Your project name>.paw** file.

Here is the list of files that you will need in order to run a TPS:

1. **<Your project name>.paw:** This is the project file that you generated with the Paws Studio.
2. **<Your project name>.obj and <Your project name>.dat:** Those files are the result of the building of the Atlas file(s), the Device Database(s), the Switch Database, and the ITA Database.
3. **<Your project name>.sig:** This file will be needed in conjunction with the **<Your project name>.swx** and the **DeviceDB.DEV** file in order to use the RTDG at run time. They are not necessary otherwise.
4. **Lex files:** The binary version of the lex files will be needed for some older versions of Atlas.
5. **Busconfi:** This file will be needed. It can be located in one of the following locations:

    a. Local to the **.paw** file. This location is known as the *Local* location.
    b. In the **<usr>\tyx\sub\<Subset>\<Station>\station**. This location is known as the *Default* location, where **<usr>** is your installation directory.

    For more information on the content of the busconfi file, please refer to the online help. Here are a few examples of busconfi content regarding the CEM channels:

- This is an example of the usage of a standard CEM dll with the default name of **Wcem.dll**:

```
"Channel"        2
```

  where 2 is the number associated with the devices that will be using the **Wcem.dll**.

- This is an example of the usage of a CEM dll called WcemDMM.dll in one of the Default location (which we address later):

```
"Channel"    2 LIB WcemDMM.dll
```

where 2 is the number associated with the devices that will be using **WcemDMM.dll**.

- This is an example of the usage of a CEM dll called WcemDMM.dll in one of the location which is not a Default location:

```
"Channel"    2 LIB .\NewFolder\WcemDMM.dll

"Channel"    3 LIB WcemScope.dll
```

  where 2 is the number associated with the devices that will be using **WcemDMM.dll**. In this case, the CEM dll WcemDMM.dll is located in a folder that is, with respect to the **.paw**, the **.obj** or the **.pax**, called **NewFolder**. Please note that this path is a relative path. You may use an absolute paths as well. The **WcemScope.dll** is on the other hand local to the **.paw**, the **.obj** or the **.pax** file.

6. **[<Your driver(s)>].dll:** In the event that you have C/C++ driver as opposed to TYX macro code drivers, you will need to have the dll(s) that were built with Paws Studio or MSVC++6.0.

   - If the busconfi does not specify the location of the dll(s), it means that you will have to place this dll, or those dlls into one of the following locations:

     a. **Locally** in the project directory (where the **.paw** is located).

     b. The station subdirectory **<usr>\tyx\sub\<Subset>\<Station>\station** containing the busconfi file. For example: **c:\usr\tyx\sub\ieee716.89\paws\station**.

     c. The folder **<usr>\tyx\prg** (documented as where the pli.exe is located by default).

d. The **OS Path** specified in your environment variables (that you can easily get by typing `C:>path` in a DOS command window). This can also be modified and configured.

If the busconfi specifies the location of the dll(s) with a relative path, or an absolute path, then you need not worry about the list above. You just have to make sure that the dll with the specified name is indeed where it is supposed to be.

This information is located in Using CEM Wizard in the online help.

For third party dlls upon which the CEM dll may depend on, you need to read the documentation associated to that/those dll(s) for the location requirements of that/those dll(s).

2.3 Obj file:

When loading the TPS in the Wrts, you should look for the **<Your project name>.obj** file. The list of files that you will need in order to run a TPS. It is the same as the list above minus the **.paw** file.

The *Local* folder should in this case be the folder that includes the **.obj** file as opposed to the **.paw** file.

2.4 Pax project:

When loading the TPS in the Wrts, you should look for the **<Your project name>.pax** file. In order to build this output file from the Paws Studio, you will need to set an option in the Paws Studio in order to build it. By default, it is not selected.

Please make sure that you include all the options that you intend to use. For instance, if you want to use the RTDG, you will need to check the option for the PAX to support it.

Here is the list of files that you will need in order to run a TPS:

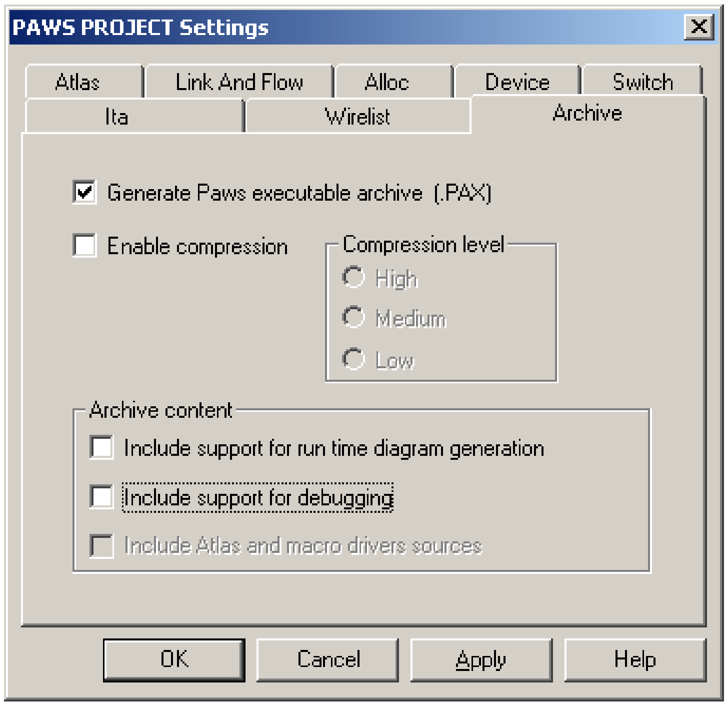
1. **<Your project name>.pax:** This is the project file that you generated with the Paws Studio. Please make sure that you checked all the options that you need to support what you intend to do with the PAX file.
2. **Busconfi:** Please refer to the comments in the Busconfi in the *Paw Project* topic above.
3. **[<Your driver(s)>].dll**: You will need the dlls in the event that you use CEM drivers. Please refer to the comments in the dll in the *Paw Project* topic above.

The PAX file may have different content depending upon the options that you have selected. There are 4 options: the basic PAX file, the PAX file with RTDG support, the PAX file with debugging capabilities and the PAX file with debugging capabilities with Atlas and device database sources. You may also chose to check all the options and have support for RTDG and debugging capabilities with sources.

Here is a description of the standalone options:

2.4.1 The basic PAX:

This is where none of the options are selected. You can use that **PAX** file with the WRTS or a client, but you do not have the ability to debug or use the RTDG. In addition, you will need the **Busconfi** file and if you have CEM drivers, all the **dlls** referred to in the Busconfi file.


PAWS PROJECT Settings

[✓] Generate Paws executable archive (.PAX)

[ ] Enable compression

Compression level: ○ High ○ Medium ○ Low

Archive content

[ ] Include support for run time diagram generation

[ ] Include support for debugging

[ ] Include Atlas and macro drivers sources

The files that are included in the PAX file are:

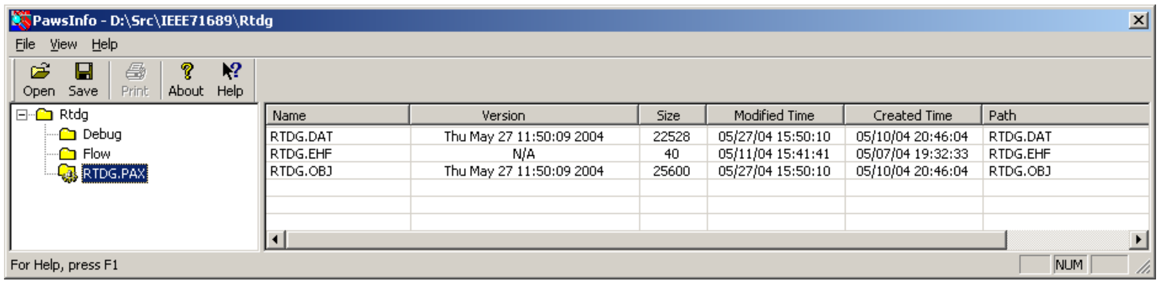
i. The **OBJ** file.

ii. The **DAT** file.

The first two include the binary information that derives from the building process of the paw project minus the files associated to the CEM driver.

The .ehf file, associated to the history of the values entered manually during runtime, is currently present but will be removed in a later version.

If you want to analyze the content of the PAX file, you can do so with the Pawsinfo.exe. Here is a sample of what you will see for this version of the PAX file:

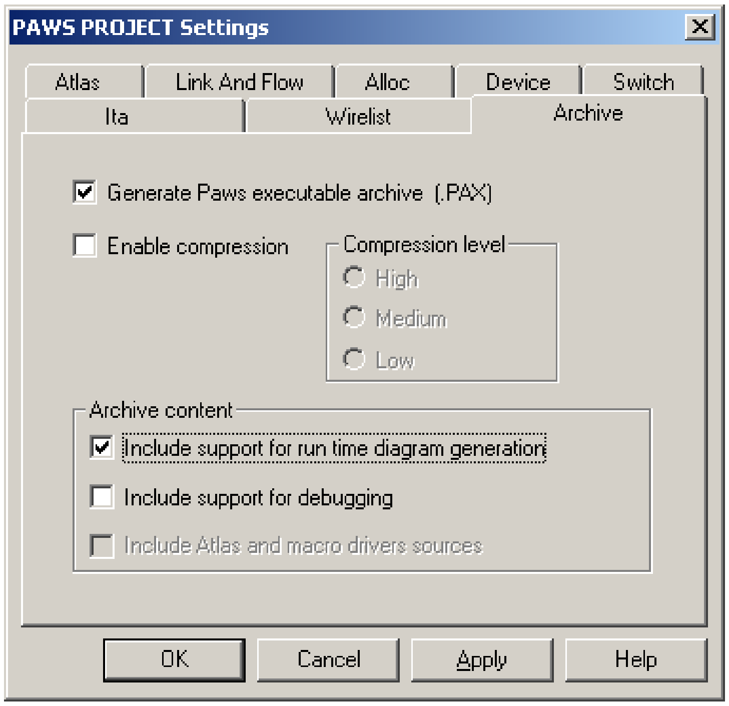


Name	Version	Size	Modified Time	Created Time	Path
RTDG.DAT	Thu May 27 11:50:09 2004	22528	05/27/04 15:50:10	05/10/04 20:46:04	RTDG.DAT
RTDG.EHF	N/A	40	05/11/04 15:41:41	05/07/04 19:32:33	RTDG.EHF
RTDG.OBJ	Thu May 27 11:50:09 2004	25600	05/27/04 15:50:10	05/10/04 20:46:04	RTDG.OBJ

For more information about Pawsinfo.exe, please refer to the online help.

2.4.2 PAX with RTDG option:

This option is to support the usage of the RTDG at runtime. You can use that **PAX** file with the WRTS or a client, and you have the ability to use the RTDG. In addition, you will need the **Busconfi** file and if you have CEM drivers, all the **dlls** referred to in the Busconfi file.


PAWS PROJECT Settings

Atlas | Link And Flow | Alloc | Device | Switch | Ita | Wirelist | Archive

[✓] Generate Paws executable archive (.PAX)

[ ] Enable compression

Compression level
( ) High
( ) Medium
( ) Low

Archive content
[✓] Include support for run time diagram generation
[ ] Include support for debugging
[ ] Include Atlas and macro drivers sources

OK | Cancel | Apply | Help

The files are that included in the PAX file are:

iii. The **OBJ** file as seen above.

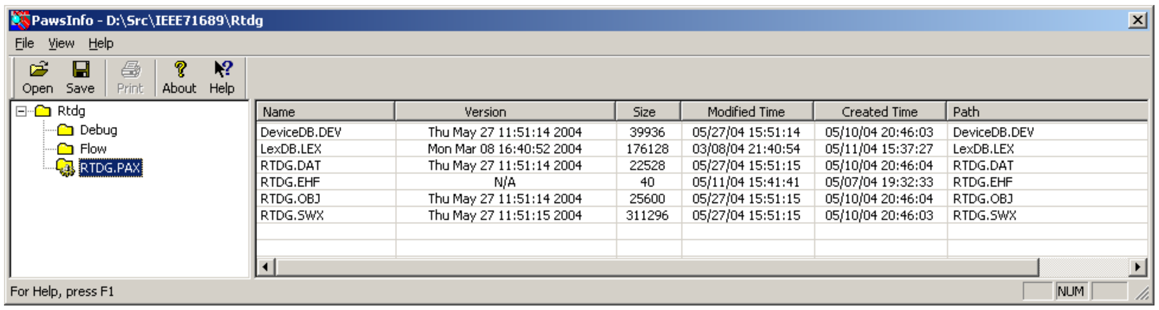
iv. The **DAT** file as seen above.

v. The **DeviceDB.DEV**.

vi. The **LexDB.LEX** file.

vii. The **SWX** file.

In order to use the RTDG at run-time, the last three files are needed because of the information they contain. If you want to analyze the content of the PAX file, you can do so with the Pawsinfo.exe. Here is a sample of what you will see for this version of the PAX file:


PawsInfo - D:\Src\IEEE71689\Rtdg

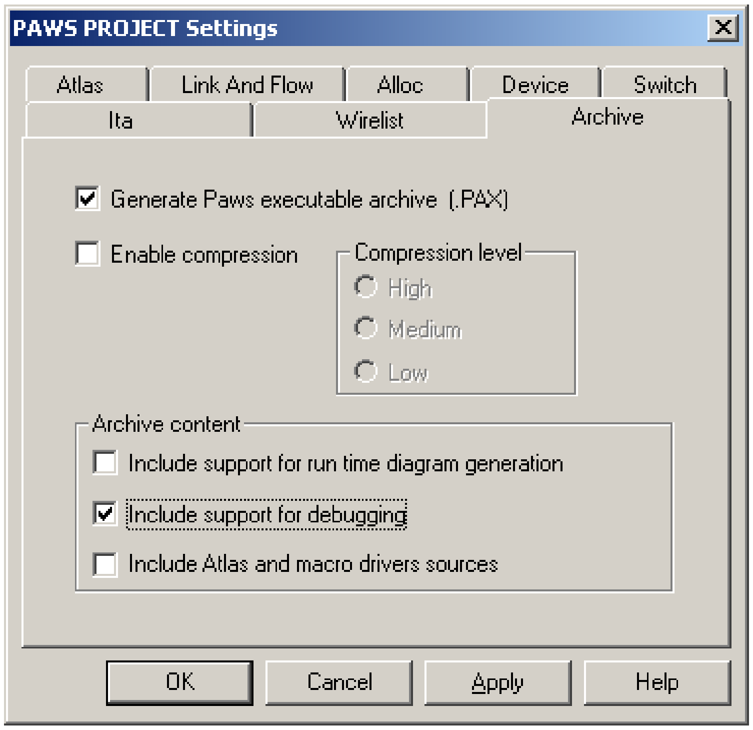
Name	Version	Size	Modified Time	Created Time	Path
DeviceDB.DEV	Thu May 27 11:51:14 2004	39936	05/27/04 15:51:14	05/10/04 20:46:03	DeviceDB.DEV
LexDB.LEX	Mon Mar 08 16:40:52 2004	176128	03/08/04 21:40:54	05/11/04 15:37:27	LexDB.LEX
RTDG.DAT	Thu May 27 11:51:14 2004	22528	05/27/04 15:51:15	05/10/04 20:46:04	RTDG.DAT
RTDG.EHF	N/A	40	05/11/04 15:41:41	05/07/04 19:32:33	RTDG.EHF
RTDG.OBJ	Thu May 27 11:51:14 2004	25600	05/27/04 15:51:15	05/10/04 20:46:04	RTDG.OBJ
RTDG.SWX	Thu May 27 11:51:15 2004	311296	05/27/04 15:51:15	05/10/04 20:46:03	RTDG.SWX

For Help, press F1

2.4.3 PAX with the debugging option but without sources:

This option is to support the RTDG and the ability to debug. This allows for the Paws Studio to launch the project and debug the Atlas and the TYX Macro code. This option is useful only if you are capable to provide the Atlas and the ddb source files so that you can see what you are debugging. It may be not desirable to include those files in the PAX file for security reason, but in the presence of the sources, this PAX file can still allow you to debug without having to rebuild the project. You can use that **PAX** file with the WRTS but you need the source files to see what you are debugging.

In addition, you will need the **Busconfi** file and if you have CEM drivers, all the **dlls** referred to in the Busconfi file.


PAWS PROJECT Settings

- [✓] Generate Paws executable archive (.PAX)
- [ ] Enable compression

Compression level
- ○ High
- ○ Medium
- ○ Low

Archive content
- [ ] Include support for run time diagram generation
- [✓] Include support for debugging
- [ ] Include Atlas and macro drivers sources

The files are that included in the PAX file are:

viii. The **OBJ** file as seen above.

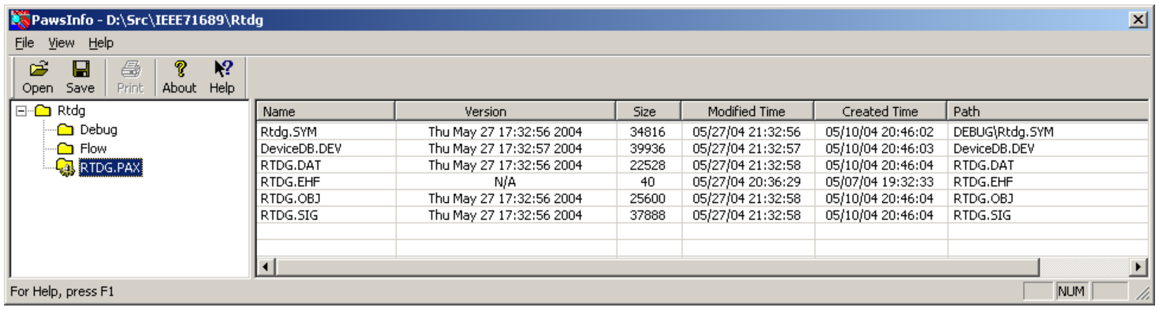
ix. The **DAT** file as seen above.

x. The **SYM** file.

xi. The **DeviceDB.DEV** file.

xii. The **SIG** file.

In order to debug the Atlas and the TYX macro code at run-time, the last three files are needed because of the information they contain. If you want to analyze the content of the PAX file, you can do so with the Pawsinfo.exe. Here is a sample of what you will see for this version of the PAX file:

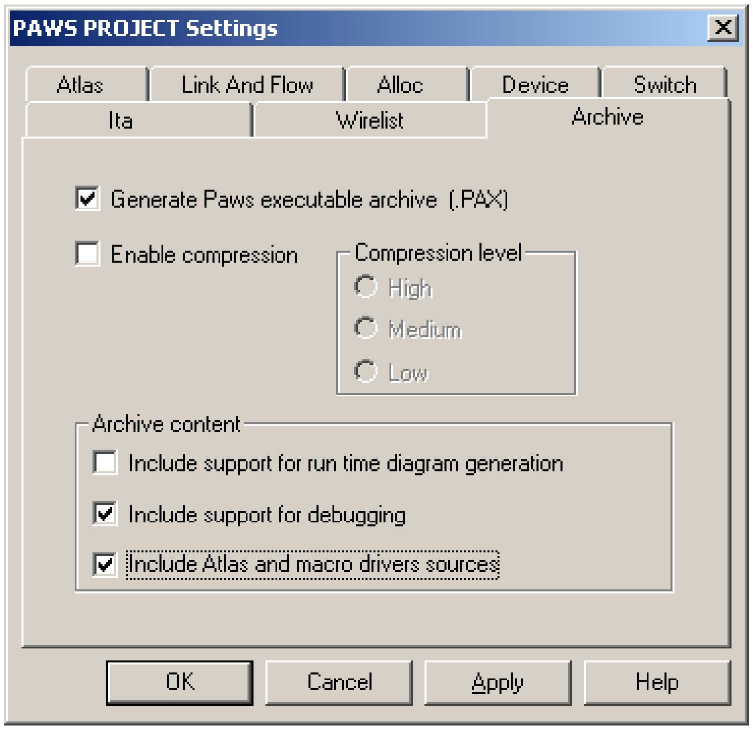


Name	Version	Size	Modified Time	Created Time	Path
Rtdg.SYM	Thu May 27 17:32:56 2004	34816	05/27/04 21:32:56	05/10/04 20:46:02	DEBUG\Rtdg.SYM
DeviceDB.DEV	Thu May 27 17:32:57 2004	39936	05/27/04 21:32:57	05/10/04 20:46:03	DeviceDB.DEV
RTDG.DAT	Thu May 27 17:32:56 2004	22528	05/27/04 21:32:58	05/10/04 20:46:04	RTDG.DAT
RTDG.EHF	N/A	40	05/27/04 20:36:29	05/07/04 19:32:33	RTDG.EHF
RTDG.OBJ	Thu May 27 17:32:56 2004	25600	05/27/04 21:32:58	05/10/04 20:46:04	RTDG.OBJ
RTDG.SIG	Thu May 27 17:32:56 2004	37888	05/27/04 21:32:58	05/10/04 20:46:04	RTDG.SIG

2.4.4 PAX with Debugging option with sources:

These options are to support the ability to debug and to have access to the source files. Please note that you can only include the sources if you select the option for debugging support. This option is useful in the context of debugging the TPS and TYX Macro code from a client such as the Litedebugger provided with the distribution or one that you created. Along with the PAX file, you don't need to provide the sources because they are included in the PAX. You can use that **PAX** file with the WRTS or any client.

In addition, you will need the **Busconfi** file and if you have CEM drivers, all the **dlls** refered to in the Busconfi file.


PAWS PROJECT Settings

Atlas | Link And Flow | Alloc | Device | Switch | Ita | Wirelist | Archive

[✓] Generate Paws executable archive (.PAX)

[ ] Enable compression

Compression level
- ( ) High
- ( ) Medium
- ( ) Low

Archive content
- [ ] Include support for run time diagram generation
- [✓] Include support for debugging
- [✓] Include Atlas and macro drivers sources

OK | Cancel | Apply | Help

The files are that included in the PAX file are:

xiii. The **OBJ** file as seen above.

xiv. The **DAT** file as seen above.

xv. The **SYM** file.

xvi. The **DeviceDB.DEV** file.

xvii. The **SIG** file.

xviii. The **KWD** files.

xix. The **LexDB.LEX** file.

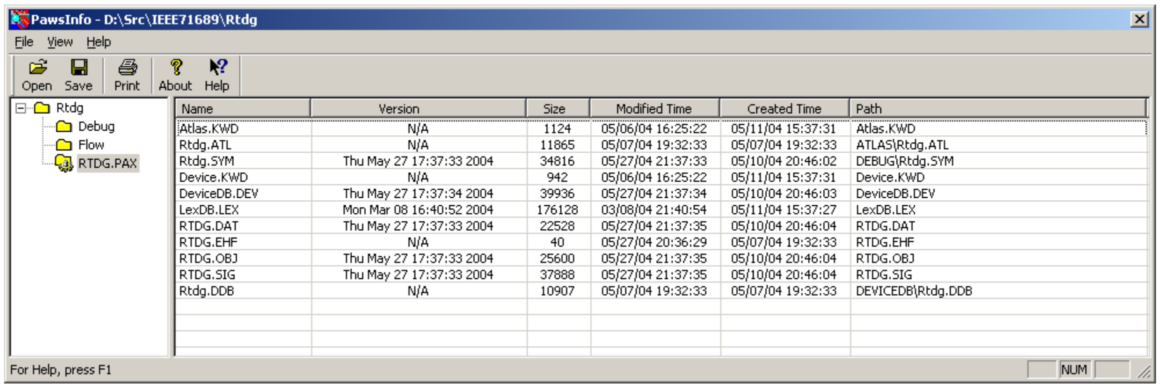
xx. The **ATL** file(s).

xxi. The **DDB** file(s).

In order to have chroma coding with the source files, the **KWD** files are necessary. Those files are part of your usual Paws Developer environment.

In addition, the **lex** file is needed. You will also find the sources files for the Atlas and the device database(s).

If you want to analyze the content of the PAX file, you can do so with the Pawsinfo.exe. Here is a sample of what you will see for this version of the PAX file:



Name	Version	Size	Modified Time	Created Time	Path
Atlas.KWD	N/A	1124	05/06/04 16:25:22	05/11/04 15:37:31	Atlas.KWD
Rtdg.ATL	N/A	11865	05/07/04 19:32:33	05/07/04 19:32:33	ATLAS\Rtdg.ATL
Rtdg.SYM	Thu May 27 17:37:33 2004	34816	05/27/04 21:37:33	05/10/04 20:46:02	DEBUG\Rtdg.SYM
Device.KWD	N/A	942	05/06/04 16:25:22	05/11/04 15:37:31	Device.KWD
DeviceDB.DEV	Thu May 27 17:37:34 2004	39936	05/27/04 21:37:34	05/10/04 20:46:03	DeviceDB.DEV
LexDB.LEX	Mon Mar 08 16:40:52 2004	176128	03/08/04 21:40:54	05/11/04 15:37:27	LexDB.LEX
RTDG.DAT	Thu May 27 17:37:33 2004	22528	05/27/04 21:37:35	05/10/04 20:46:04	RTDG.DAT
RTDG.EHF	N/A	40	05/27/04 20:36:29	05/07/04 19:32:33	RTDG.EHF
RTDG.OBJ	Thu May 27 17:37:33 2004	25600	05/27/04 21:37:35	05/10/04 20:46:04	RTDG.OBJ
RTDG.SIG	Thu May 27 17:37:33 2004	37888	05/27/04 21:37:35	05/10/04 20:46:04	RTDG.SIG
Rtdg.DDB	N/A	10907	05/07/04 19:32:33	05/07/04 19:32:33	DEVICEDB\Rtdg.DDB