Adapting Usage to Different Environments and Needs

Insure++ can be easily adapted to different environments and needs. This section describes some of the ways Insure++ can be customized. In this section:

Instrument Using Multiple Processes

If you are using a machine with a CPU that has multiple cores and/or hyperthreading, Insure++ can take advantage of this by instrumenting with multiple processes. Insure++ can create an instrumentation process for each available logical CPU. You can also specify the number of processes to create. Depending on the makeup of your project and the machine hardware, this can significantly reduce instrumentation time.

Visual Studio users who build using multiple processes with the `/MP` flag can take advantage of this without any additional changes. Insure++ will automatically recognize the flag and pass it along to the compiler once instrumentation has finished.

To enable instrumentation with multiple processes, use the `/MP` or `-Zmp` flag on your command line.

To specify number of processes, use `/MPn` or `-Zmpn`, where *n* is the number of desired processes. Insure++ will not create more processes than available logical CPUs. If you specify more processes than are available, Insure++ will still only use the maximum available. Here is a chart of the flags:

Flag	Effect
/MP, -Zmp	Insure++ will detect maximum number of logical CPUs and use all available for instrumentation.
/MPn, -Zmpn	Insure++ will detect the maximm number of logical CPUs and instrument using *n* processes or the maximum number of processes if *n* is greater than the max.

Examples

For an Intel Core i7 quad-core CPU with Hyperthreading (8 logical CPUs):

- /MP = 8 processes for instrumentation
- -Zmp6 = 6 processes for instrumentation
- -Zmp3 = 3 processes for instrumentation
- /MP12 = 8 processes for instrumentation

Code Coverage Warnings

If your sources are on a network share and you use Insure++'s code coverage, multiple processes for instrumentation can cause problems when writing to the coverage file as a result of the inability to obtain a lock. If you experience coverage file corruption, you can either disable coverage, disable multiple processes, or use sources on a local drive.

If your sources are on a local drive and you still experience coverage file corruption, you can increase the number of retries for Insure++ to acquire a lock on the coverage file by using the `uselockf` configuration. See Configuration Options (psrc) for more information.

COM Objects, Windows Services, and Child Processes

Some applications must be invoked in atypical ways. For example, Windows services may be executed from the service control manager. ActiveX controls may be loaded into other executables spawned automatically. Child processes may be executed by calls to CreateProcess from parent processes.

In these cases, the Image File Execution Options can be used to run the program through InsureSpy. To use Image File Execution Options, perform the following steps:

1. Open the Insure++ Control Panel and click the the **Advanced** tab
2. Enter your executable name (without a path) in the Image File Execution Options: Executable field.
3. Enter a value from the following table in the Debugger field:

`devenv.exe`	Runs Insure++ integrated with Microsoft debugger in Visual Studio 2003 and higher. The executable will contain instrumented components, so you should also enabled Automatic InsureSpy Integrated Debugging. See General and InsureSpy.
`inject.exe /inject_x64.exe`	Runs Insure++ integrated with the InsureSpy debugger. This option is best for console builds using the command line.

`insureSpy.exe` `/insureSpy.exe`	Runs Insure++ integrated with the InsureSpy debugger. This option is best for GUI builds in Windows environments.
`msdev.exe`	Runs Insure++ integrated with the Microsoft debugger in Visual Studio 6. The executable will contain instrumented components, so you should also enabled Automatic InsureSpy Integrated Debugging. See General and InsureSpy.

The Image File Execution Options mechanism is part of the Windows operating system. Insure++ provides access to this mechanism found in the Insure++ Control panel for your convenience. If you prefer, you can alter the registry keys directly at `HKEY_LOCAL_MACHINE/Software/Microsoft /WindowsNT/ CurrentVersion/"Image File Execution Options"/`.

When you are done debugging the executable with Insure++, be sure to remove the executable name from the list, or the .exe file specified in the Debugger field will be invoked every time that executable runs.

When debugging a Windows service with Insure++, allow the service to interact with the desktop by enabling the service option in the Services Control Manager (**Start> Settings> Control Panel> Administrative Tools> Services> /services/> Log On** tab). Enable the **Log on as: Local System account** option and the allow service to interact with desktop box. Because the service will be running as a system process, you must have [Insure++_Install_ Dir] /bin on your system PATH environment variable. It is not sufficient to have it on your user PATH environment variable.

If you are using the Image File Execution Options on a service, special care may need to be taken to ensure that these options work. In particular, if the key `HKLM>SYSTEM>ControlSet001/Services/ Service_Name/Image Path` contains a path with spaces in it, you must ensure that there are quotation marks around the path.

See the service_demo.txt file in the [Insure++ Install dir]/examples directory for an example of running Insure++ on a service application.

You cannot allow a Windows service to interact with the desktop if you are not connected to the console (such as, using Terminal Services or Remote Desktop). The option in the Services Microsoft Management Console (MMC) may suggest that the service can interact with the desktop, but it will not actually interact with the desktop that you are looking at. If this situation occurs, Inject or InsureSpy will fail to connect to Insra and the following error will open:

“The service did not start or control request in a timely fashion."

To work around this, use the /console flag to Remote Desktop (mstsc.exe /console) to connect to the console of the remote machine.

Precompiled Headers

Insure++ supports the use of precompiled headers. However, the nature of its support differs from Microsoft’s support. The most significant difference is that Insure++ requires that no tokens other than preprocessor directive tokens be placed before the header stop point in the primary source file. For example:

```
unsupported Stdafx.cpp:
------------------------
int x;
#include "Stdafx.h"
------------------------
```

To configure such a file to build with Insure++, move the `int x` declaration into the `Stdafx.h` header. Other conventions that Insure++ does not support include:

- The /YX command line flag for automatic pch processing.
- The use of #pragma hdrstop "filename".

Insure++ does support the use of #pragma hdrstop (without naming a pch file), as well as the naming of the header stop point using /Y[cu]<headername>.

If you want Insure++ to support your precompiled headers, we recommend that you have a single header contain everything to be precompiled. This header should be included at the top of every file that depends on the precompiled header.

If you want Insure++ to perform strict checking for the valid use of precompiled headers, use the psrc option `pchcheck` to enable this checking. By default, this strict checking is not performed.

Insure++ will alert you to unsupported pch configurations by reporting an error with the error code PCH_ERROR. If a project uses unsupported pch configurations, you will need to disable pch (to prevent the application from using pch) in order to build with Insure++. You can disable pch automatically with the psrc option `pchdisable`. However, some projects will not build–even without Insure++–if pch is disabled. In these cases, first fix the configurations to build without pch optimization. After they are fixed, try to build them with Insure++ again.

InsureSpy

This feature enables you to get runtime error-detection from Insure++ while running in the Microsoft Debugger (msdev.exe). This feature is also available for Visual Studio 2003 and higher (devenv.exe).

When Insure++ is installed, running an instrumented executable (an .exe file that is built with Insure++) through the Visual Studio debugger will automatically invoke InsureSpy. No special settings are required for this behavior. When running a non-instrumented executable through the debugger, the default behavior is that InsureSpy is NOT invoked, that is, the debugger operates as it did before Insure++ was installed.

Sometimes it is desirable to have Visual Studio invoke InsureSpy on non-instrumented executables. For example:

- When minimal error detection is desired without rebuilding the application and integrated debugging is desired.
- When the instrumented module is a component such as a DLL or OCX which is loaded into a non-instrumented executable and integrated debugging is desired.

In these cases, the integrated InsureSpy can be invoked by selecting Automatic InsureSpy Integrated Debugging in the General tab of the Insure++ Control Panel. When this option is enabled, all executables that are started in the debugger will be run through the integrated InsureSpy. Another way to run an executable (instrumented or otherwise) through InsureSpy is to simply click the Insure++ Debug icon in the Insure++ toolbar.

This option does not affect running InsureSpy outside of the debugger; for example, running Insure++ by clicking Insure++ Execute, selecting **File> Run** in Insra, or calling InsureSpy or inject explicitly from the command-line or from Image File Execution Options, as described in Using Image File Execution Options.

The following table lists command line options accepted by InsureSpy:

InsureSpy Command Line Option	**Description**

`-children`	Causes InsureSpy to debug all processes that descend from the debuggee. For example, if parent.exe creates child.exe, and child.exe is instrumented, the following example would debug both processes. If `-children` is not specified, InsureSpy will execute child processes as normal (non-debugged).
	
	Examples:
	
	`InsureSpy.exe -children parent.exe`
	
	`InsureSpy_x64.exe -children parent.exe`
	
	All instrumented processes should be debugged with InsureSpy

Setting Breakpoints

When using the InsureSpy Integrated Debugging feature, you can set a breakpoint at `_Insure_trap_error` under **Edit> Breakpoints** in Visual Studio 6 or **Debug> New Breakpoint** in Visual Studio 2003 and higher. This will cause the debugger to hit a breakpoint whenever Insure++ finds an error, so you can examine variables and single-step through with the debugger.