What is BoostC

BoostC is our latest generation C compiler that works with PIC18, PIC16 and some PIC12 processors. BoostC was designed to rival Hi-Tech C compiler. This ANSI C compatible compiler addresses limitations that exist in our C2C-plus compiler. It supports features like source level debugging, signed data types, structures/unions and pointers. The BoostC compiler will eventually replace the C2C-plus compiler.

Check the BoostC compiler Reference Manual in PDF format here.

The BoostC compiler can be used with the SourceBoost IDE (Integrated Development Environment), or can be integrated into Microchip MPLAB.
BoostC works under MS Windows 95/98/ME/NT/2000/XP.

Take a look at the BoostC Compiler Reference Manual.

What's in the software package

When you download the SourceBoost software package you get the SourceBoost IDE, BoostC compiler and all the other SourceBoost compilers. The compilers and IDE are activated by using the appropriate activation key.

License Limitations

All license options for the BoostC 'C' compiler support PIC12, PIC16 and PIC18 Target Devices. Please check the supported devices list to confirm the devices you require are supported.

Upgrade Policy

All licenses get free minor revision upgrades. That means any BoostC V6.x license can be upgraded to the latest BoostC V6.x at zero cost.

Major revision upgrades are chargeable. That means you will have to pay an upgrade fee in order to upgraded BoostC V6.x license to a BoostC V7.x license.

The **Full License** has unlimited RAM and ROM usage. Suitable for projects of all shapes and sizes. The definite choice of the serious programmer. Non-commercial use.

The **Pro License** has unlimited RAM and ROM usage. The correct choice for the professional programmer. This license allows commercial use.

Supported Device List

Want to know if a desired target device is supported by the BoostC 'C' Compiler, then checkout the list below.

If a device you want to use is not it the list, send us an email and we will see what we can do!

The Supported Device List

PIC12

PIC12C671, PIC12CE673, PIC12CE674, PIC12F609, PIC12HV609, PIC12F615, PIC12HV615, PIC12F629, PIC12F635, PIC12C672, PIC12F675, rfPIC12F675h, rfPIC12F675k, rfPIC12F675f, PIC12F683

PIC16

PIC16CE623, PIC16CE624, PIC16CE625, PIC16C432, PIC16C433, PIC16C554, PIC16C557, PIC16C558, PIC16C61, PIC16F616, PIC16HV616, PIC16C62, PIC16C62A, PIC16C62B, PIC16CR62, PIC16C620, PIC16C620A, PIC16CR620A, PIC16C621, PIC16C621A, PIC16C622, PIC16C622A, PIC16F627, PIC16F627A, PIC16F628, PIC16F628A, PIC16C63, PIC16C63A, PIC16CR63, PIC16F630, PIC16F631, PIC16F636, PIC16F639, PIC16C64, PIC16C64A, PIC16CR64, PIC16C642, PIC16F648A, PIC16C65, PIC16C65A, PIC16C65B, PIC16CR65, PIC16C66, PIC16C662, PIC16C67, PIC16F676, PIC16F677, PIC16F684, PIC16F685, PIC16F687, PIC16F688, PIC16F689, PIC16F690, PIC16C71, PIC16C710, PIC16C711, PIC16C712, PIC16C715, PIC16C716, PIC16F716, PIC16C717, PIC16C72, PIC16C72A, PIC16CR72, PIC16F72, PIC16C73, PIC16C73A, PIC16C73B, PIC16F73, PIC16F737, PIC16C74, PIC16C74A, PIC16C74B, PIC16F74, PIC16C745, PIC16F747, PIC16C76, PIC16F76, PIC16C765, PIC16F767, PIC16C77, PIC16F77, PIC16C770, PIC16C771, PIC16C773, PIC16C774, PIC16F777, PIC16C781, PIC16C782, PIC16F785, PIC16HV785, PIC16F818, PIC16F819, PIC16CR83, PIC16F83, PIC16C84, PIC16CR84, PIC16F84, PIC16F84A, PIC16F87, PIC16F870, PIC16F871, PIC16F872, PIC16F873, PIC16F873A, PIC16F874, PIC16F874A, PIC16F876, PIC16F876A, PIC16F877, PIC16F877A, PIC16F88, PIC16F883, PIC16F884, PIC16F886, PIC16F887, PIC16F913, PIC16F914, PIC16F916, PIC16F917, PIC16C923, PIC16C924, PIC16C925, PIC16C926, PIC16F946

PIC18

PIC18F1220, PIC18F1230, PIC18F1320, PIC18F1330, PIC18F2220, PIC18F2221, PIC18F2320, PIC18F2321, PIC18F2331, PIC18F2410, PIC18F24J10, PIC18C242, PIC18F242, PIC18F2420, PIC18F2423, PIC18LF2423, PIC18F2431, PIC18F2439, PIC18F2450, PIC18F2455, PIC18F248, PIC18F2480, PIC18F2510, PIC18F25J10, PIC18F2515, PIC18C252, PIC18F252, PIC18F2520, PIC18F2523, PIC18LF2523,

PIC18F2525, PIC18F2539, PIC18F2550, PIC18F258, PIC18F2580, PIC18F2585, PIC18F2610, PIC18F2620, PIC18F2680, PIC18F2682, PIC18F2685, PIC18F4220, PIC18F4221, PIC18F4320, PIC18F4321, PIC18F4331, PIC18F4410, PIC18F44J10, PIC18C442, PIC18F442, PIC18F4420, PIC18F4423, PIC18LF4423, PIC18F4431, PIC18F4439, PIC18F4450, PIC18F4455, PIC18F448, PIC18F4480, PIC18F4510, PIC18F45J10, PIC18F4515, PIC18C452, PIC18F452, PIC18F4520, PIC18F4523, PIC18LF4523, PIC18F4525, PIC18F4539, PIC18F4550, PIC18F458, PIC18F4580, PIC18F4585, PIC18F4610, PIC18F4620, PIC18F4680, PIC18F4682, PIC18F4685, PIC18C601, PIC18F6310, PIC18F63J11, PIC18F6390, PIC18F63J90, PIC18F6410, PIC18F64J11, PIC18F6490, PIC18F64J90, PIC18F65J10, PIC18F65J11, PIC18F65J15, PIC18F6520, PIC18F6525, PIC18F6527, PIC18C658, PIC18F6585, PIC18F65J90, PIC18F66J10, PIC18F66J15, PIC18F6620, PIC18F6621, PIC18F6622, PIC18F6627, PIC18F66J60, PIC18F66J65, PIC18F6680, PIC18F67J10, PIC18F6720, PIC18F6722, PIC18F67J60, PIC18C801, PIC18F8310, PIC18F83J11, PIC18F8390, PIC18F83J90, PIC18F8410, PIC18F84J11, PIC18F84J15, PIC18F8490, PIC18F84J90, PIC18F85J10, PIC18F85J11, PIC18F85J15, PIC18F8520, PIC18F8525, PIC18F8527, PIC18C858, PIC18F8585, PIC18F85J90, PIC18F86J10, PIC18F86J15, PIC18F8620, PIC18F8621, PIC18F8622, PIC18F8627, PIC18F86J60, PIC18F86J65, PIC18F8680, PIC18F87J10, PIC18F8720, PIC18F8722, PIC18F87J60, PIC18F96J60, PIC18F96J65, PIC18F97J60

Optimizing Compiler and Linker features

SourceBoost IDE Integration

The BoostC compiler integrates seamlessly into the MS Windows SourceBoost IDE (Integrated Development Environment).

Base Data types:

- 8 bit char - signed and unsigned.
- 16 bit int - signed and unsigned.
- 32 bit long - signed and unsigned.

Special Data types:

- Single bit - single bit data type for efficient flag storage.
- Fixed address - fixed address data types allow easy access to target device registers.
- Read only code memory based variables.

Code Producing Features:

- ANSI 'C' compatible - Makes code highly portable.
- Produces optimized code for both PIC16 (14bit core) and PIC18 (16bit core) targets.

- Support for Data Structures and Unions - Data structures and arrays can be comprised of base data types or other data structures. Arrays of base data types or data structures can be created.
- Support for pointers - pointers can be used in "all the usual ways".
- Inline Assembly - Inline assembly allows hand crafted assembly code to be used when necessary.
- Inline Functions - Inline functions allows a section of code to be written as a function, but when a reference is made to it the inline function code is inserted instead of a function call. This speeds up code execution.
- Eliminates unreachable (or dead) code - reduces code memory usage.
- Removal of Orphan (uncalled) functions - reduces code memory usage.
- Minimal Code Page switching - code where necessary for targets with multiple code pages.
- Automatic Banks Switching for Variables - allows carefree use of variables
- Efficient RAM usage - local variables in different code sections can shared memory. The linker analyzes the program to prevent any clashes.
- Function pointers (not available yet).
- Dynamic memory management.
- Output file in Intel Hex format.

Debugging features:

- Source Level and Instruction Level Debugger - linker Generates COF file output for source level debugging under SourceBoost Debugger.
- Step into, Step over, Step out - functions operate at source level and instruction level.
- Multiple Execution Views - see where the execution of the code is at source level and assembly level at the same time.
- Monitoring variables - variables can be added to the watch windows to allow there values to be examined. There is no need to know where a variable is stored.

Librarian:

- Allows generations of library files - this simplifies management and control of regularly used code.
- Reduce compilation time - using library files reduces compilation time.

Code Analysis:

- Call tree view - in SourceBoost IDE displays the function call tree.
- Target Code Usage - From the complete program, down to Function level the code space usage can be view in SourceBoost IDE.
- Target RAM Usage - From the complete program, down to Function level the RAM usage can be view in SourceBoost IDE.

C language superset

BoostC compiler has some advanced features "borrowed" from C++ language. These features allows to develop more flexible and powerful code but one doesn't have to use them.

References

References can be used as function arguments. This provides very effective way to change values of the variables passed into a function as references:

```
void foo( char& x )  //'x' is a reference
{
  x = 10;
}

void main( void )
{
  char a = 1;
  foo( a );  //before the call 'a' is 1 and after the call it becomes 10
  ...
}
```

Function overloading

Functions can have same names and differ only by number and type of their arguments:

```
void foo( void )  //'foo' number 1
{
...
}

void foo( char *ptr )  //'foo' number 2
{
...
}

void foo( char a, char b )  //'foo' number 3
{
...
}

void main( void )
{
  foo();  //'foo' number 1 gets called
  foo( "test" );  //'foo' number 2 gets called
  foo( 10, 20 );  //'foo' number 3 gets called
  ...
}
```

Function templates

Functions can be declared and defined using data type placeholders. This feature allows writing very general code (for example linked lists) that isn't tightened to a particular data type and what may be more important to create libraries in form of header files:

```
template <class T>
void foo( T *t )
{
...
}

void main( void )
{
  short s;
  foo<char>( "test" );  //'foo( char* )' gets called
  foo<short>( &s );  //'foo( short* )' gets called
  ...
}
```