# Welcome to Orange C IDE help

Orange C IDE is an integrated development environment tailored to use with the Orange C tool set that supports C and C++ development and debugging.

Orange C IDE supports most functionality of the tool set; however it is not a standalone program.  To accomplish its tasks it usually spawns an appropriate tool from the tool set, such as the compiler or linker.  Output from spawned tools is integrated into the information window.

Orange C IDE may also be used as a standalone editor.

The following topics contain general information to help you get started:

Getting Started

Orange C IDE has the following general functions:

- Configurable Integrated editor
- Project window, with integrated make facility
- Resource editor
- Information windows
- Error List
- Variable Usages Window
- Integrated debugger
- Debugging windows
- Find and Replace
- Bookmarks
- Integrated platform help
- Browse information
- Printing
- Menus
- Toolbars
- Hot keys
- General Properties Dialog
- Project Properties Dialog

# For licensing information, look here:

GNU Version 3 License

# Getting Started

Orange C IDE is a workarea - based system.  That means to do anything significant other than simply editing files (such as compiling or debugging) a workarea must first be set up.  The workarea has information about **Projects**, which are executable or DLL files that are to be created.  A list of sources files is associated with each project; when attempting to create the project the IDE will apply the appropriate tool to each source file to compile it, then link the results together to create the project.  Orange C IDE maintains a list of dependency files and automatically rebuild all related source files when one of the dependency files changes.

The list of source files may be any combination of C/C++ language files, Assembly language files, RC files, and libraries.  Orange C IDE will automatically apply the compiler, the assembler, the resource compiler, or the linker depending on the file name extension of the source file.  If a **Make** option is selected, only files which have changed, or whose dependencies have changed, will be recompiled.  If the **Build All** option is selected Orange C IDE will rebuild all files irregardless of what has changed.

Creating a Workarea is a four-step process.  First, select **File -> Work Area -> New Workarea** from the menu bar.  A New Work Area Dialog  prompts for the name of the workarea.  By default, this dialog will want to save your workarea in a new subdirectory under the **My Documents** folder, but this location can be changed.

After creating the Workarea, a project must be created.  To create a project select the **File -> Work Area -> Add -> New Project** menu item from the menu bar.  A New Project Dialog prompts for the name and type of the project.  Type in the name of a project and select its type.  The project name should not have a file extension, as one will be added based on the type you select.  Normally, one would select a **Console** project for standard C programming, or text-based programs that are meant to be be accessed from a command prompt.  **Windows GUI** may be selected to make a program that is intended to have a windowing user interface.  The other project types are used for sharing code between projects.

Once the project is created, right-clicking on the project's name or on the source header will cause the Project Context Menu  to appear.   Press **Add -> New File** to create a new file to type in.  Later, when you attempt to save the file an Open File Dialog prompts you for the name to save it as.

If this is your first program and you have selected **Console** as the program type you might try typing something like:

```
#include <stdio.h>

int main(int argc, char *argv[])
{
printf("Hello World");
}
```

This short program is a classic first program that simply displays the text Hello World when you run it.

Selecting **File -> Work Area -> Save** from the menu bar will ensure the workarea and project is saved to disk.

At this point the project may be built by selecting one of the items on the Build Menu.  Once it is built it may be debugged by starting the Integrated Debugger from either the Debug Menu or the Debug Toolbar.

GNU GENERAL PUBLIC LICENSE
Version 3, 29 June 2007

Preamble

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works.  By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program--to make sure it remains free software for all its users.  We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors.  You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price.  Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights.  Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received.  You must make sure that they, too, receive or can get the source code.  And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software.  For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so.  This is fundamentally incompatible with the aim of protecting users' freedom to change the software.  The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable.  Therefore, we have designed this version of the GPL to prohibit the practice for those products.  If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary.  To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow.

## TERMS AND CONDITIONS

0. Definitions.

"This License" refers to version 3 of the GNU General Public License.

"Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

"The Program" refers to any copyrightable work licensed under this License.  Each licensee is addressed as "you".  "Licensees" and "recipients" may be individuals or organizations.

To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy.  The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

A "covered work" means either the unmodified Program or a work based on the Program.

To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy.  Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To "convey" a work means any kind of propagation that enables other parties to make or receive copies.  Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

"Target code" means any artifact generated by the Orange C compiler toolchain in the course of processing input files.

"Independent modules" means any file processed by the Orange C compiler toolchain while generating Target Code.

The "Runtime Library" means the portions of the C/C++ libraries not covered by external licensing, when processed into artifacts to be used while generating Target Code.

An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License.  If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.

The "source code" for a work means the preferred form of the work for making modifications to it.  "Object code" means any non-source form of a work.

A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The "System Libraries" of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form.  A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities.  However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work.  For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

You have permission to propagate a work of Target Code formed by combining the Runtime Library with Independent Modules, even if such propagation would otherwise violate the terms of GPLv3. You may then convey such a combination under terms of your choice, consistent with the licensing of the Independent Modules.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users' Legal Rights From Anti-Circumvention Law.

No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies.

You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

a) The work must carry prominent notices stating that you modified it, and giving a relevant date.

b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices".

c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.

d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.

b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.

c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.

d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.

e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

"Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or

b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or

c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or

d) Limiting the use for publicity purposes of names of licensors or authors of the material; or

e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or

f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

### 8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

### 9. Acceptance Not Required for Having Copies.

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

### 10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An "entity transaction" is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

### 11. Patents.

A "contributor" is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's "contributor version".

A contributor's "essential patent claims" are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, "control" includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a "patent license" is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To "grant" such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. "Knowingly relying" means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others' Freedom.

If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License.

Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

<one line to give the program's name and a brief idea of what it does.>
Copyright (C) <year>  <name of author>

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <https://www.gnu.org/licenses/>.

Also add information on how to contact you by electronic and paper mail.

If the program does terminal interaction, make it output a short notice like this when it starts in an interactive mode:

<program>  Copyright (C) <year>  <name of author>
This program comes with ABSOLUTELY NO WARRANTY; for details type `show w'.
This is free software, and you are welcome to redistribute it
under certain conditions; type `show c' for details.

The hypothetical commands `show w' and `show c' should show the appropriate parts of the General Public License. Of course, your program's commands might be different; for a GUI interface, you would use an "about box".

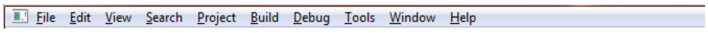
You should also get your employer (if you work as a programmer) or school, if any, to sign a "copyright disclaimer" for the program, if necessary. For more information on this, and how to apply and follow the GNU GPL, see <https://www.gnu.org/licenses/>.

The GNU General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License. But first, please read <https://www.gnu.org/licenses/why-not-lgpl.html>.

# Menus

The IDE has two types of menus. The first type is the menu bar at the top of the program's window interface. The second type is context-sensitive menus that are available when right-clicking in the area of various special windows.

The menu bar at the top of the program appears as follows:



It has the following menus:

- File menu Shows file, workarea, and general menu items
- Edit Menu Shows items related to editing
- View Menu Shows the states of special windows, browse information, and bookmarks
- Search Menu Shows options related to searching for text
- Project Menu Shows options related to the active project
- Build Menu Shows items related to compiling and linking programs
- Debug Menu Shows items related to debugging
- Tools Menu Shows user defined tools and items related to customizing the IDE
- Resource Menu Shows items related to resource editor windows
- Window Menu Shows items related to window management
- Help Menu Shows items related to documentation

The following additional context-sensitive menus are available by right-clicking on various windows in the resource editor:

- Accelerators Context Menu Shows menu items for the Accelerator edit window
- Dialog Context Menu Shows menu items for the Dialog edit window
- Image Context Menu Shows menu items for the Bitmap, Cursor, and Icon edit windows
- Menu Context Menu Shows menu items for the Menu edit window
- RCData Context Menu Shows menu items for the RCData edit window
- String Table Context Menu Shows menu items for the Strings edit window
- Version Context Menu Show smenu items for the Version editor edit window

The following additional context-sensitive menus are available by right-clicking on various windows:

- Client Context Menu Shows items related to managing editor windows
- Editor Context Menu Shows items related to editing, bookmarks, and debugging
- Information Context Menu Shows items related to compiling and linking programs
- Watch Context Menu Shows items related to the watch window
- ~~Breakpoints Context Menu~~ Shows items related to the breakpoints window
- Memory Context Menu Shows items relates to size of memory units
- Workarea Context Menu Shows items related to the workarea
- Project Context Menu Shows Items related to the selected project
- Folder Context Menu Shows items related to the selected folder
- File Context Menu Shows items related to the selected file

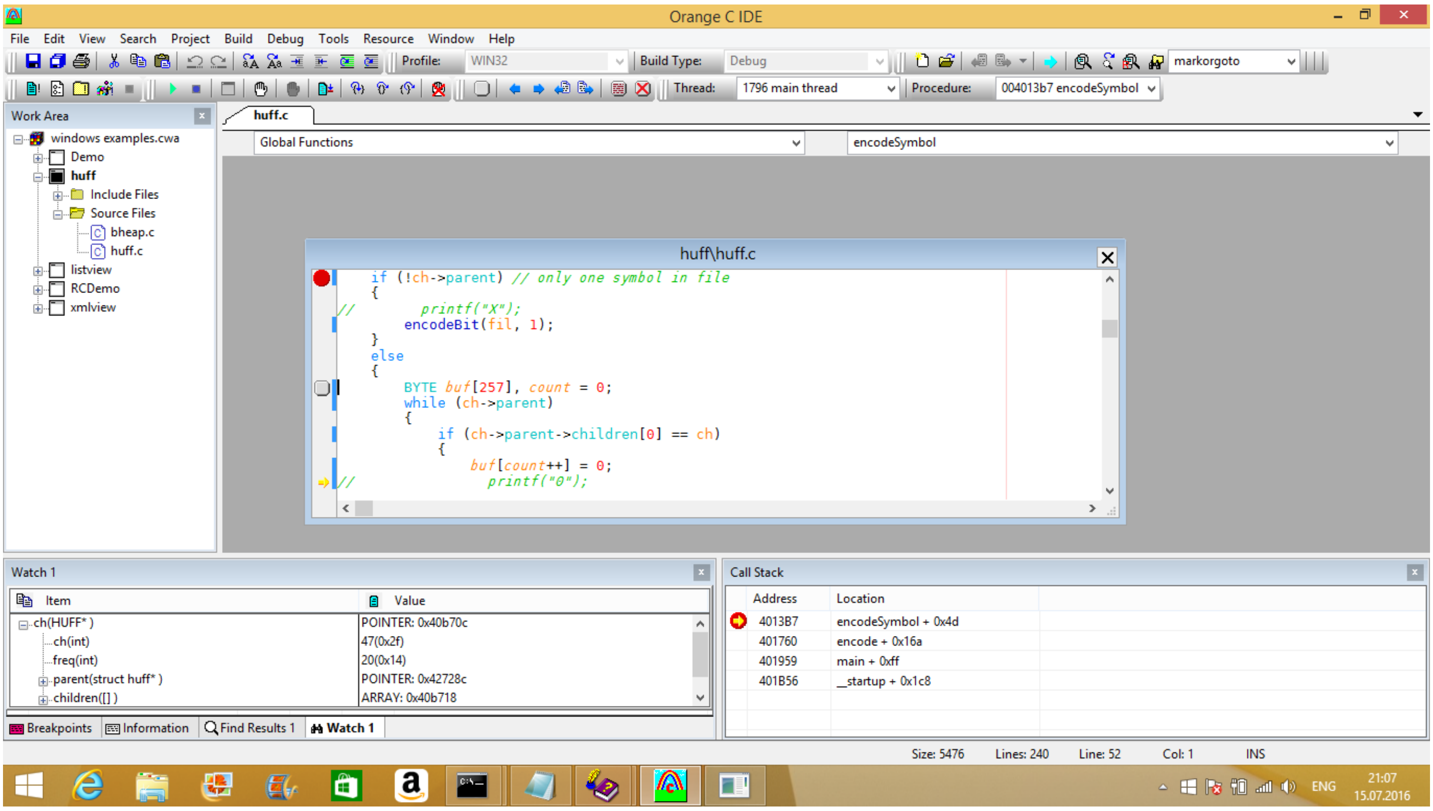
# Editor Window

The Editor Window iss accessible by opening a file, either by selecting FILE->FILE->OPEN from the Main <enu or by clicking on the file in the Workarea Window.

The editor is a colorizing editor with basic functionality similar to a rich edit control, and various extended functionality added to support various IDE functionality. It can colorize for the C language, NASM, and Resource files.

Colorization is done based on file extension. Configuration of basic functionality is supported.

The editor window appears similar to what follows:



```
    if (!ch->parent) // only one symbol in file
    {
//          printf("X");
        encodeBit(fil, 1);
    }
    else
    {
        BYTE buf[257], count = 0;
        while (ch->parent)
        {
            if (ch->parent->children[0] == ch)
            {
                buf[count++] = 0;
//                printf("0");
```

Watch 1

| Item | Value |
|---|---|
| ch(HUFF* ) | POINTER: 0x40b70c |
| ch(int) | 47(0x2f) |
| freq(int) | 20(0x14) |
| parent(struct huff* ) | POINTER: 0x42728c |
| children([] ) | ARRAY: 0x40b718 |

Call Stack

| Address | Location |
|---|---|
| 4013B7 | encodeSymbol + 0x4d |
| 401760 | encode + 0x16a |
| 401959 | main + 0xff |
| 401B56 | __startup + 0x1c8 |

In this example, the Integrated Debugger has been started. The blue bar to the left of the text shows code lines which have generated code associated with them, and are suitable for breakpoints. The red circle is a break point. The yellow arrow shows the position at which the program flow was stopped. This example also shows a bookmark, which is a grey square.

When one or more files are opened for editing, a series of tabs at the top of the client area will show a list of all open files, and allow easy navigation to a given file. If there are two many open files, an arrow button to the far right of the tabs will allow opening a menu to select from available files. A star next to a file name indicates the file has changed and needs to be saved.

Above all the editor windows but below the tab with the file names is the Jump List. The box at the left selects a type of variable such as a global functions or global variables, and the box at the right gives a list of function or variable names. Selecting a function or variable name from the box at the right opens a window for the associated file (if it isn't already open) and positions the cursor on the line where the variable or function is declared.

Other ways to navigate to an open window is to view the window list on the Window menu, and to use the Client Context Menu to open the Select Window dialog.

Various menu items are available to help manage the editor windows. These include items on the File menu of the main menu, on the Edit menu of the main menu, the Windows menu, the Client Context Menu, and the Context menu available in edit windows.

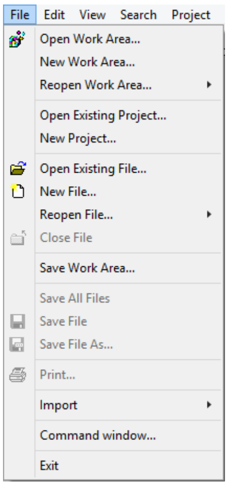
Double clicking in the area to the left of the window will toggle a Breakpoint wit the selected line.

Pressing SHIFT-F1 while an edit window is selected will bring up C runtime library help for the word under the cursor.

In the background, the IDE will parse through files that are in open edit windows, to get lists of type information and structure elements. This information is used for code completion, and to show hints about variable types.

# File Menu

The File Menu has menu items related to files and some generic commands. These include the file and workarea submenus. It appears similar to what follows:



**Open Work Area** is used to browse to a work area to open

**New Work Area** creates a new Work Area

**Reopen Work Area** opens a menu which shows recent Work Areas that have been opened. Clicking on one opens it.

**Open Existing Project** brings an existing project into the work area

**New Project** creates a new project in the work area

**Open Existing File** browses to a source file to open

**New File** creates a new source file

**Reopen File** opens a menu which shows recent source files which have been opened.  Clicking on one opens it.

**Close File** closes the current edit window

**Save Work Area** saves the Work Area and Projects if they have changed

**Save All Files** saves changes to all open source files

**Save File** saves changes to the source file which is currently being edited

**Print** is used to print files. It opens the standard windows printer dialog and allows selection of various properties related to the print request.

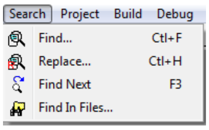
**Import** is used to bring old CC386 workspaces and projects forward.

**Command Window** opens a command prompt.  The command prompt will have sufficient environment variables defined to build the project.

**Exit** exits the IDE.  If any files need to be saved or the debugger needs to be closed, a prompt will appear requesting confirmation.

## Search Menu

The Search Menu shows options related to searching for text, and replacing one text with another.  It appears similar to what follows:



**Find** brings up a Find/Replace dialog configured for locating text in editor windows.

**Replace** brings up a Find/Replace dialog configured for replacing text in editor windows.

**Find Next** locates the next occurrance of the last text searched for.

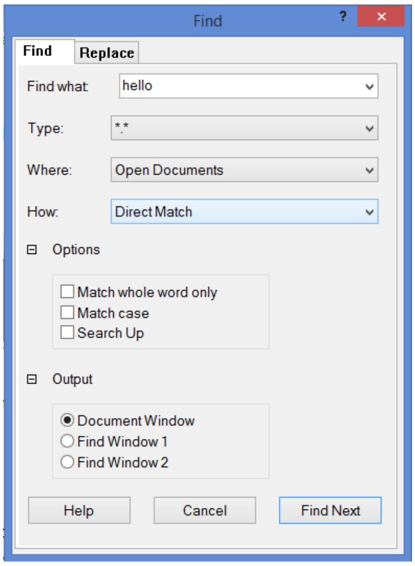
**Find In Files** brings up a Find/Replace dialog configured for locating text in files that are not in editor windows.

## Find/Replace Dialog

The Find/Replace dialog is accessible by clicking one of the items on the Edit Toolbar, by selecting one of the items from the Search Menu , or by pressing **CTRL-F** or **CTRL-H**.

The Find/Replace dialog is a generic dialog with two tabs.  One tab is used for locating text, and the other tab is used for replacing text.  The text may be searched for in a variety of places, including in open windows, as part of a project or workarea, or on the disk.  Text that is found may either be found one item at a time with navigation to each item, or a batch of locations may be sent to one of two windows dedicated to showing text that has been located.

When used to locate text, the Find/Replace dialog appears similar to what follows:



Find what: hello

Type: *.*

Where: Open Documents

How: Direct Match

⊟ Options

□ Match whole word only
□ Match case
□ Search Up

⊟ Output

● Document Window
○ Find Window 1
○ Find Window 2

Note that some items will sometimes be closed to make the dialog smaller.

**Find What** allows selection of the text to locate

**Type** allows selection of a filter to be used for file names, by default all files are searched but the scope may be narrowed to various types of files the IDE understands if necessary.

**Where** allows selection of what documents are in the scope of the search.  This may be one of the following

**Current document** any text in the window that currently has focus
**Open documents** any text in any open window
**Current project** any text in any file within the selected project
**All projects** any text in any file that exists in the workarea
**Orange C include path** any text in system include files
**User specified path** any text in a user-specified path.

**Path** allows selection of a path to look in for files, when Where is set to User Specified Path.  The button at the right will bring up a window that allows selection of the path, or the path may be directly typed in.  Otherwise this window doesn't appear.

**How** specifies how matches will be calculated.

**Direct Match** characters must directly match
**Regular Expressions** regular expressions may be used to determine the match
**Wildcard** characters generally directly match, however **?** may be used to match any character and ***** may be used to match any sequence of characters

**Options** allow for various special case behaviors while searching.  They include

**Match Whole Word** when selected, any text that is matched must be the same word as **Find What -** no matches will be made on partial words
**Match Case** when selected, any text that is matched must be the same case as specified in  **What**
**Recursive** when selected in conjunction with a **User Specified Path**, will cause the search to extend to subdirectories
**Search Up** when selected, the search will be done toward the beginning of the documents instead of the end.

**Output** selects what is going to happen when something is located.

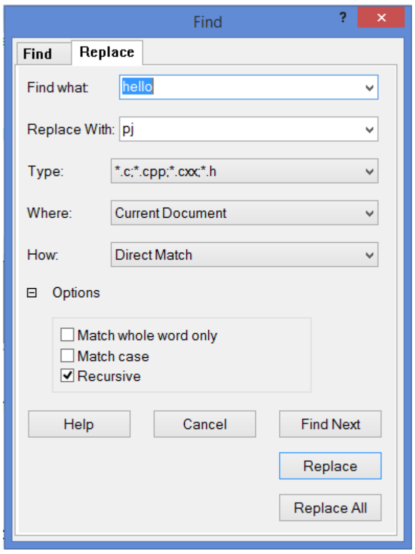
**Document Window** navigate to the next location found and then stop locating text
**Find Window 1** show the location in Find Window 1 and continue locating text
**Find Window 2** show the location in Find Window 2 and continue locating text

**Find Window 1** and **Find Window 2** may be selected from tabs on the Information Window

Note that for each type of location that may be specified in **Where**, options and output selections will be remembered independently.

There are two find windows, find-related information may be sent to either window.

When used to replace text, the Find/Replace dialog appears as follows:



Find what: hello

Replace With: pj

Type: *.c;*.cpp;*.cxx;*.h

Where: Current Document

How: Direct Match

⊟ Options

☐ Match whole word only
☐ Match case
☑ Recursive

Note that some items will sometimes be closed to make the dialog smaller.

**Find What** allows selection of the text to locate

**Replace With** allows selection of the text to replace it with

**Type** allows selection of a filter to be used for file names, by default all files are searched but the scope may be narrowed to various types of files the IDE understands if necessary.

**Where** allows selection of what documents are in the scope of the search.  This may be one of the following

**Current document** any text in the window that currently has focus
**Open documents** any text in any open window
**Current project** any text in any file within the selected project
**All projects** any text in any file that exists in the workarea
**Orange C include path** any text in system include files
**User specified path** any text in a user-specified path.

**Path** allows selection of a path to look in for files, when Where is set to User Specified Path.  The button at the right will bring up a window that allows selection of the path, or the path may be directly typed in.  Otherwise this window doesn't appear.

**How** specifies how matches will be calculated.

**Direct Match** characters must directly match
**Regular Expressions** regular expressions may be used to determine the match
**Wildcard** characters generally directly match, however **?** may be used to match any character and ***** may be used to match any sequence of characters

**Options** allow for various special case behaviors while searching.  They include

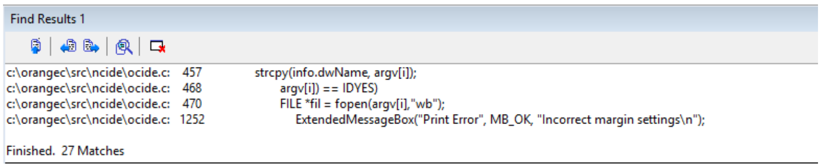
**Match Whole Word** when selected, any text that is matched must be the same word as **Find What -** no matches will be made on partial words
**Match Case** when selected, any text that is matched must be the same case as specified in **Find What**
**Recursive** when selected in conjunction with a **User Specified Path**, will cause the search to extend to subdirectories

When performing a single replace, the IDE will navigate to the location that is found and show a single replacement.  When doing a Replace All, all replacements will be made by loading the files selected by 'where' into editor windows as needed, and making the changes without saving them.  The changes may then be reviewed before being accepted...

Note that for each type of location that may be specified in **Where**, options and output selections will be remembered independently.

## Find Windows

The find windows serve as targets for find and replace.



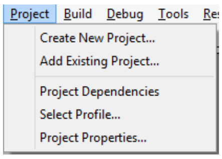
```
c:\orangec\src\ncide\ocide.c:   457          strcpy(info.dwName, argv[i]);
c:\orangec\src\ncide\ocide.c:   468                argv[i] == IDYES);
c:\orangec\src\ncide\ocide.c:   470                FILE *fil = fopen(argv[i],"wb");
c:\orangec\src\ncide\ocide.c:   1252                   ExtendedMessageBox("Print Error", MB_OK, "Incorrect margin settings\n");

Finished.  27 Matches
```

In the example the search was for the word 'arg'. It found a number of matches which are listed in the window. Clicking on a match takes one to the line of source code for the match.

The various icons at the top of the window allow navigating in the window, and searching for text in the window.

## Project Menu

The Project Menu has items related to projects. It appears similar to what follows:



**Create New Project** brings up the New Project Dialog to allow entry of parameters for creating a new project.

**Add Existing Project** brings up an Open File Dialog to allow locating an existing project to load into the current workarea.

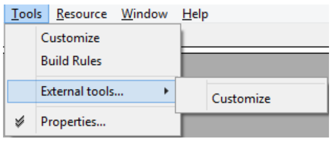
**Project Dependencies** brings up the Project Dependencies dialog to allow selection of which projects depend on other projects.

**Select Profile** opens the Select Profile dialog to allow selection of the current profile

**Project Properties** opens the project properties dialog for the active project.

## Tools Menu

The tools menu has options for external tools, and for configuring the IDE. It appears similar to what follows:



In this case an external tool which has been labeled 'my tool' appears on the **External Tools** submenu. When selected, this will perform a specific function which has been configured through the Customize option on the **External Tools** submenu. Other external tools may be configured or removed through this option.

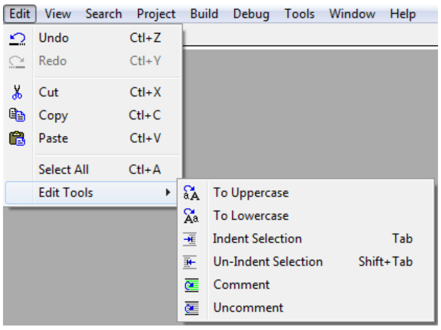
**Customize** (on the tools menu) brings up the Toolbar Customization dialog which allows configuration of which toolbars will be shown, as well as the configuration of what buttons will be shown and in what order.

**Build rules** brings up the Build Rules dialog, which allows customization of build rules.

**Properties** brings up the General Properties page, which allows configuration of the IDE.

## Edit Menu

The Edit Menu has items which allow browsing and manipulating the text in the edit windows. It appears similar to what follows:



**Cut, Copy** and **Paste** are used to move selected text to and from the clipboard.

**Undo** undoes recently made edits. The Undo buffer is fairly deep. Note that you can undo changes even after you save a file, until the file window is closed.

**Redo** redoes changes that have recently been undone.

**Select All** selects all the text in the current edit window.

**Uppercase** and **Lowercase** change the selected text to either upper or lower case.

**Indent** moves the selected right text by one tab space. This may uses either spaces or tabs depending on the Editor Formatting Configuration

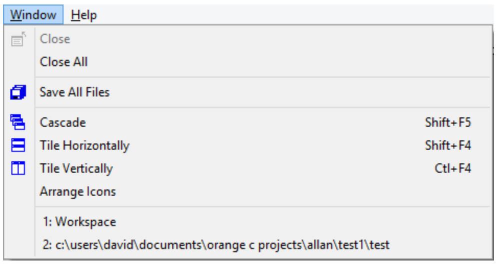
**Unindent** is similar to **indent**, except it moves the text left.

**Comment** adds comment markers to the beginning of each line in a selected block of text, to make the text inactive code.

**Uncomment** removes comment markers to the beginning of each line in a selected block of text, to make the text active code again.

## Window Menu

The Window Menu has various items used for controlling windows in the client area of the IDE. It appears similar to what follows:



**Close** closes the currently selected window.

**Close All** closes all edit windows on the client area, but not special windows that have been undocked.

**Save All Files** saves changes to the files in all open windows.

**Cascade** orders all client area windows so that they are the same size and are overlapped in a staggered fashion, where the title bar of each successive window is just below and to the right of the previous window.

**Tile Horizontally** orders all client area windows so that they are stacked from the top of the client area to the bottom

**Tile Vertically** orders all client area windows so that they are stacked from the left of the client area to the right.

**Arrange Icons** organizes any client area windows that have been iconized.

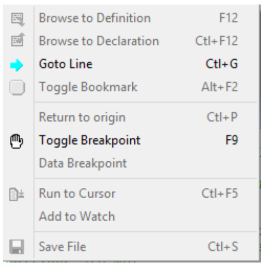
The remaining elements on this menu show some of the files that are open. The currently selected file will have a star next to it. If there are too many open files a ***More Windows...*** item will appear, which opens the Select Window Dialog. This allows navigation to any window shown in the client area.

Special Windows such as the Watch Window will also appear in the client area if undocked, and will show up in this list.

Window navigation may also be performed by using the tabs at the top of the client area.

## Editor Context Menu

The Edit Context Menu iss accessible by right-clicking in one of the Editor Windows. It has menu items which relate to editing, browsing, and debugging. It appears similar to what follows:



**Browse To Definition** uses ~~Browse Information~~ to browse to the definition of a variable or function

**Browse To Declaration** uses Browse Information to browse to the declaration of a variable or function

**Toggle Bookmark** toggles a Bookmark at the selected line

**Goto line** opens the Goto Dialog. This allows positioning on a given line when the line number is entered.

**Return To Origin** opens the window the current breakpoint is associated with and navigates to the breakpoint line.

**Toggle Breakpoint** toggles the break point setting for the current line. See Breakpoints for further information about breakpoints.

**Data Breakpoint** opens the Data Breakpoint dialog to allow adding a data breakpoint on the word under the cursor.

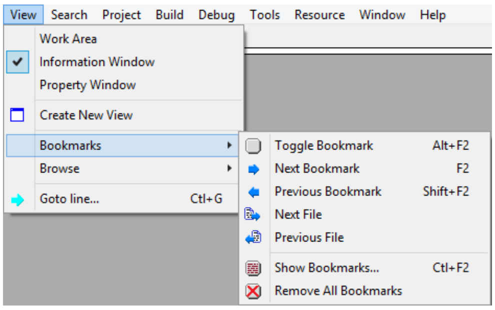
**Run To Cursor** starts the program being debugged if it is currently stopped, and keeps it running until either a is hit or the flow of execution reaches the line with the cursor.

**Add To Watch** adds the identifier under the cursor to the Watch Window and shows its value. The Watch Window will be opened if it is currently closed.

**Save File** saves the file in the edit window

## View Bookmarks Menu

The View Bookmarks Menu has menu items related to Bookmarks. It appears similar to what follows:



**Toggle Bookmark** toggles the bookmark state of the line currently selected in the editor window.

**Next Bookmark** moves the cursor to the line of the next bookmark in the bookmarks list. If necessary it opens a new edit window with the related file.

**Previous Bookmark** moves the cursor to the line of the previous bookmark in the bookmarks list. If necessary it opens a new edit window with the related file.

**Next File** moves the cursor forward past all the bookmarks left in the current file, and positions to the first bookmark in the next file in the list.

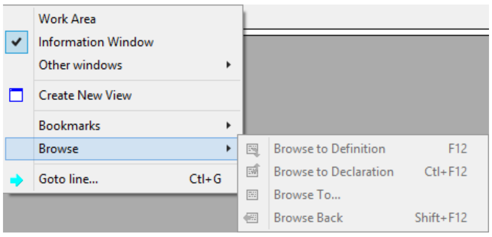
**Previous File** moves the cursor backward past all the previous bookmarks in the current file, and positions to the first bookmark in the previous file in the list.

**Show Bookmarks** opens the Bookmark Dialog. This allows the ability to browse to a specific bookmark.

**Remove All Bookmarks** removes all bookmarks from the bookmarks list.

## View Browse Menu

The View Browse Menu has menu items related to Browse Information. It appears similar to what follows:



**Browse To Definition** browses to the definition of the word under the cursor using Browse Information

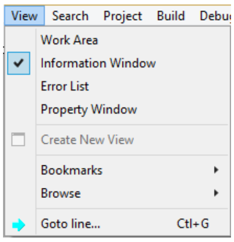
**Browse To Declaration** browses to the declaration of the word under the cursor using Browse Information

**Browse To** opens the Browse To Dialog. This allows selecting a variable name to locate the definition for.

**Browse Back** undoes the last **Browse** or **Browse To** command.

## View Menu

The View Menu has various items which allow for viewing windows or positioning to bookmarks and browsing. It appears similar to what follows:



**Workarea** opens or closes the Workarea Window. When the window is displayed a check mark will appear next to the menu item.

**Information Window** opens or closes the Information Window. When the window is displayed a check mark will appear next to the menu item.

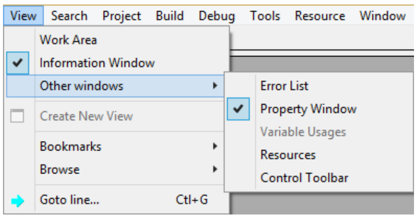
**Create New View** makes a copy of an existing editor window. The new window will be a new view that looks at the same data as the existing view. Changes in one view will be reflected in the other. This mechanism allows one to view two separate locations in the same file.

**Bookmarks** opens the View Bookmarks Menu.

**Browse** opens the View Browse Menu.

**Goto Line** opens the Goto Line dialog. This allows selection of a line number to navigate to..

**View Other** opens a submenu that looks similar to what follows:



On this menu,

**Error List** opens or closes the Error List Window. When the window is displayed a check mark will appear next to the menu item.

**Property Window** opens or closes the Resourced Property Window. This window shows information about resources being edited.

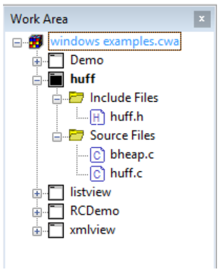
**Variable Usages** opens or closes the Variable Usages Window. This window uses browse information to show information about the usages of a variable or function, such as where it is declared and defined, and where it is used.

**Resources** opens or closes the Resource Window.

**Control Toolbar** opens the Dialog Control Toolbox , which chooses controls to place on a dialog.

## Workarea Window

The Workarea Window iss accessible by clicking Work Area from the View Menu. It holds a list of projects to be created. It appears similar to what follows:



Each project is an executable, DLL, or static library file.  Executables are programs that can be run. DLLs are like executables except they hold shared code that executables can access.  Static libraries hold code that is not itself executable but can be combined with other code to make an executable.

When there are multiple projects, one project will always be selected as the active project.  This is the project that will be used by default when an action such as an attempt to debug occurs.

Underneath each project is a list of all the source files which are required to create the project.  When a build is requested the IDE parses through the list of files to see which ones have changed, and spawns an appropriate compiler for each one.  For file types it knows about such as C language files, the IDE will automatically parse the files to make a list of dependencies.

New projects may be created in the workarea using the Workarea Menu or the Workarea Context Menu.  There are also commands to import pre-existing projects.

Once a project is created, a few tree items will appear underneath it.  These items hold the source files for the project.  By using the Workarea Context Menu, new files can be inserted underneath each header.  Additionally, folders may be added to organize the files.  Folders and files may be dragged around to change the heirarchy.

Note that the folders are not related to how data is stored in the file system.

When adding files to the project, the IDE will attempt to classify them and put them in one of the stock folders, which are **Source Files**, **Include Files**,  and **Resource Files**.  If the associated folder has been removed it will be added again to have a place to put the file.  Once files are added they can be moved to another folder within the project.

Note that the **Include Files** folder is a convenience that allows one to access include files from the IDE.  The IDE will automatically calculate the dependencies of source files for purposes of the build process.  This is unrelated to the contents of the **Include Files** folder, which must be manually populated if it is desired to have any files there.

After a file is added to the project, double clicking on a file or selecting it and pressing enter will open that file in an editor window.

If a project is a library file, other projects will automatically be linked against it.

The workarea file stores various general settings related to the user interface, and a reference to a separate configuration file for each project.  Each project may appear in multiple workareas.  The paths stored in the configuration files are relative.  That means that projects may be moved around on the disk as long the workareas and projects remain in the same relative location to each other.

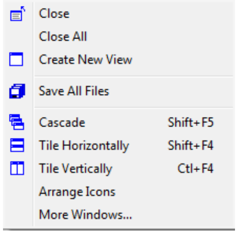
The arrow keys can be used to navigate through this menu.  The ENTER key opens the file the cursor is over.  CTL-INSERT and CTL-DELETE can be used to open and close sub trees.

INSERT and DELETE can be used to add and remove projects and files.  INSERT opens the Open File Dialog to allow selection of file names or new files to add.  DELETE will only display a warning if an attempt is made to remove a project; removing a file succeeds silently

Any project or file that is removed from the workarea still exists on the disk, and can be added again at a later time.

## Client Context Menu

The Client Context Menu can be reached by right-clicking in the client area, outside the context of one of the editing windows.  It has items related to the client area such as tiling and closing/saving files.  It appears similar to what follows:



**Close** closes the currently selected window.

**Close All** closes all edit windows on the client area, but not special windows that have been undocked.

**Save All** saves the contents of all open windows.

**Create New View** makes a copy of an existing editor window.  The new window will be a new view that looks at the same data as the existing view.  Changes in one view will be reflected in the other.  This mechanism allows one to view two separate locations in the same file.

**Cascade** orders all client area windows so that they are the same size and are overlapped in a staggered fashion, where the title bar of each successive window is just below and to the right of the previous window.

**Tile Horizontally** orders all client area windows so that they are stacked from the top of the client area to the bottom

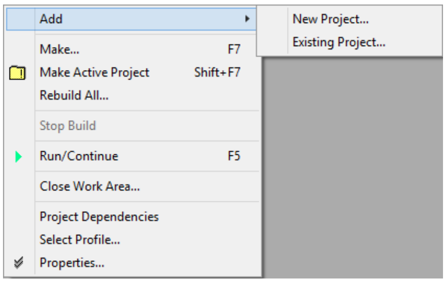
**Tile Vertically** orders all client area windows so that they are stacked from the left of the client area to the right.

**Arrange Icons** organizes any client area windows that have been iconized.

**More Windows...** opens the Select Window Dialog to allow a window to be selected.

## Workarea Context Menu

The Workarea Context Menu may be reached by right-clicking in the WorkArea Window on the name of the workarea.  It appears similar to what follows:



**New Project** brings up the New Project dialog for selecting the characteristics of a new project.

**Existing Project** brings up the Open File dialog for importing an existing project.

**Make** calculates which files have changed, and recompiles those files as well as recreates any files which depend on them

**Make Active Project** performs a make, but only on the active project.

**ReBuild All** rebuilds all projects in the workarea by removing all output files then doing a **make**.

**Stop Build** stops a build which is in progress.

**Run/Continue** starts the debugger if it isn't running, at the active project. Debugging Windows are opened at this point. The program will stop at the first break point it encounters, or at the start of the *main()* or *WinMain()* function if no breakpoints exist.

**Close Work Area** closes the workarea.

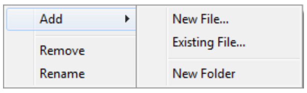
**Project Dependencies** brings up the Project Dependencies dialog to allow selection of which projects depend on other projects.

**Select Profile** opens the Select Profile dialog to allow selection of the current profile

**Properties** opens the Project Properties Dialog

## Folder Context Menu

The folder context menu iss accessible by right-clicking on a folder in the Workarea Window. has items related to creating and removing folders within a project. It appears similar to what follows:



**New File** opens an Open File dialog to allow entering the name of a new file which is being created as it is added to the project.

**Existing File** opens an Open File dialog to allow opening an existing file to add it to the project.

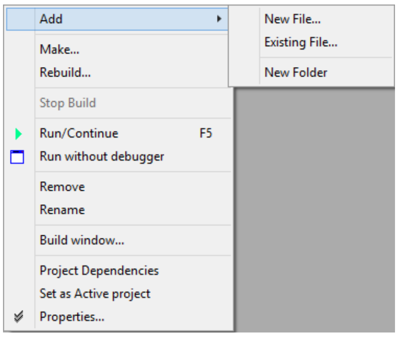
**New Folder** creates a new folder underneath the current one.

**Remove** removes a folder. Any files in the folder will be moved to the parent of the removed folder.

**Rename** allows renaming the folder.

## Project Context Menu

The Project Context Menu iss accessible by right-clicking on a project name in the Workarea Menu. It has menu items related to the project. It appears similar to what follows:



**New File** opens a New File Dialog dialog to allow entering the name of a new file which is being created as it is added to the project.

**Existing File** opens an Open File dialog to allow opening an existing file to add it to the project.

**New Folder** creates a new folder underneath the current one.

**Make** calculates which files have changed, and recompiles those files as well as recreates any files which depend on them

**ReBuild** rebuilds all projects in the workarea by removing all output files then doing a **make**.

**Stop Build** stops a build which is in progress.

**Run/Continue** starts the debugger if it isn't running, at the active project. Debugging Windows are opened at this point. The program will stop at the first break point it encounters, or at the start of the *main()* or *WinMain()* function if no breakpoints exist.

**Run Without Debugger** starts the program as a separate executable, without debugging. If it is a console program, a separate console window will be created for it to display data in.

**Remove** removes a project from the workarea. The project and all its files will still reside on the file system.

**Rename** allows renaming the project.

**Build Window** is an alternate way to build the project that performs the build using a make file, in a separate console window.  Errors shown in this window will not be readily accessible for locating corresponding lines in source files.

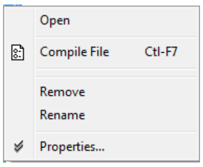
**Project Dependencies** brings up the Project Dependencies dialog to allow selection of which projects depend on other projects.

**Set As Active Project** sets the current project as the active project, which selects it as the default project the debugger will use.

**Properties** opens the project properties window

## File Context Menu

The File Context Menu may be reached by right clicking on an individual source file in the Workarea Window.  Ithas menu items related to individual files within a project.  It appears similar to what follows:



**Open** opens an Edit Window with the contents of the file.  In the case of a resource, it opens the Resource Window for the file, it is then necessary to click on a resource to edit it.

**Compile File** uses the appropriate tool to compile, assemble, or otherwise process the file to create an output file.

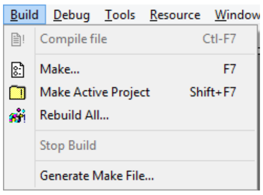
**Remove** removes a file from the project.  The file will still reside on the file system.

**Rename** allows renaming the file.

**Properties** opens a property window for editing build-specific properties for the file.

## Build Menu

The Build Menu has menu items related to compiling and linking programs.  It appears similar to what follows:



**Compile File** compiles the file in the currently active editor window.

**Make** calculates which files have changed, and recompiles those files as well as recreates any files which depend on them

**Make Active Project** performs a make, but only on the active project.

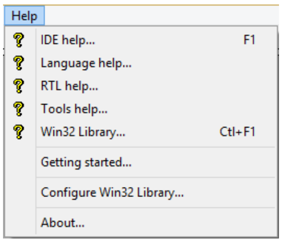
**ReBuild All** rebuilds all projects in the workarea by removing all output files then doing a **make**.

**Stop Build** stops a build which is in progress.

**Generate Make File** creates a Generated Make File.

## Help Menu

The Help Menu has menu items related to documentation.  It appears similar to what follows:



**IDE Help** shows the index page of this help file.

**Language Help** shows help about the C Language.

**RTL Help** shows help specific to the C Run-time Library.

**Tools Help** shows help specific to the Command Line Tools, and help related to compiler extensions used by the C compiler in this package.

**WIN32 Help** opens the selected WIN32 help system to its main page.

**Getting Started** shows general information about how to get started with creating a project in this Ide

**Configure WIN32 Library** downloads the WIN32 offline help library, when the help mode is set to Windows 8

**About** shows the version information of the IDE.

## Generated Make File

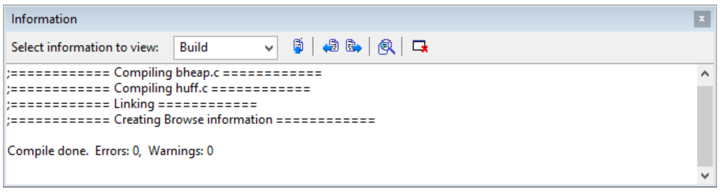
The IDE is capable of converting a workarea into a make file compatible with the OMAKE command line make utility.   This allows exporting the build steps, so that the file can be built from the command line instead of the IDE.

The generation is selected through the **Generate Make File** option on the Build Menu. If the workarea has multiple project, the make file will contain scripts suitable for rebuilding all projects.

The paths in the make file are made relative to the location of the make file.  This means that if projects are moved to a new location, they must be kept in the same relative location to each other and to the make file.

## Information Window

The Information window is accessible by clicking **Information Window** on the View Menu.  It has a combo box to select what type of information to view.



Select information to view: Build

```
;============ Compiling bheap.c ============
;============ Compiling huff.c ============
;============ Linking ============
;============ Creating Browse information ============

Compile done.  Errors: 0,  Warnings: 0
```

In this example, we see the IDE creating the output files for a project.

The combobox  will generally auto-select itself when information is being displayed in them, or they can be manually selected at any time.

The various icons at the top of the window allow navigation and searching in the text.

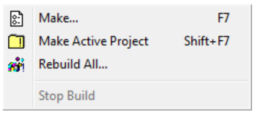
The **Build** tab gives information about the compile in progress.  Errors and general status can be viewed here.  Clicking on a line that shows an error or warning will open an edit window and position the file to that line.  In the build tab, errors will be in red and warnings will be in blue.

The **Debug** tab gives information about the debug status, such as what DLLs and THREADs are starting or ending.  It also displays messages from the program that are sent via the OutputDebugString API function.  These messages will be in blue.

Right-Clicking on the information window pane opens the Information Context Menu.

## Information Context Menu

The Information Context Menu is accessed by right-clicking in the Information Window. It has menu items to aid in rebuilding the current project.  It appears as follows:



**Make** calculates which files have changed, and recompiles those files as well as recreates any files which depend on them

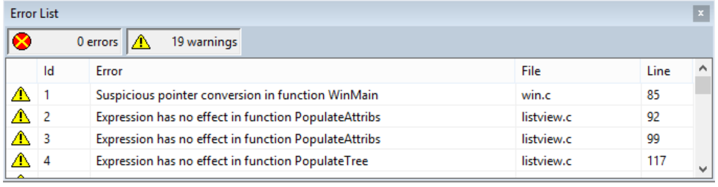
**Make Active Project** performs a make, but only on the active project.

**ReBuild All** rebuilds all projects in the workarea by removing all output files then doing a **make**.

**Stop Build** stops a build which is in progress.

## Error List

The Error List is accessible by clicking **Error List** on the View Menu.  This window shows a list of errors and warnings.   It appears as follows:



| | Id | Error | File | Line |
|---|---|---|---|---|
| ⚠ | 1 | Suspicious pointer conversion in function WinMain | win.c | 85 |
| ⚠ | 2 | Expression has no effect in function PopulateAttribs | listview.c | 92 |
| ⚠ | 3 | Expression has no effect in function PopulateAttribs | listview.c | 99 |
| ⚠ | 4 | Expression has no effect in function PopulateTree | listview.c | 117 |

In this example, we see several warnings (yellow icon).   If there were an error, the icon next to it would show a red circle with an X.

The buttons at the top of the window allow configuration of whether to view errors, warnings, or both.   Clicking on an error line brings up an Editor Window to view the line with the problem.

The contents of this window are derived from the Information Window's listing of build results, however, the Information Window does not have to be displayed for this window to function.

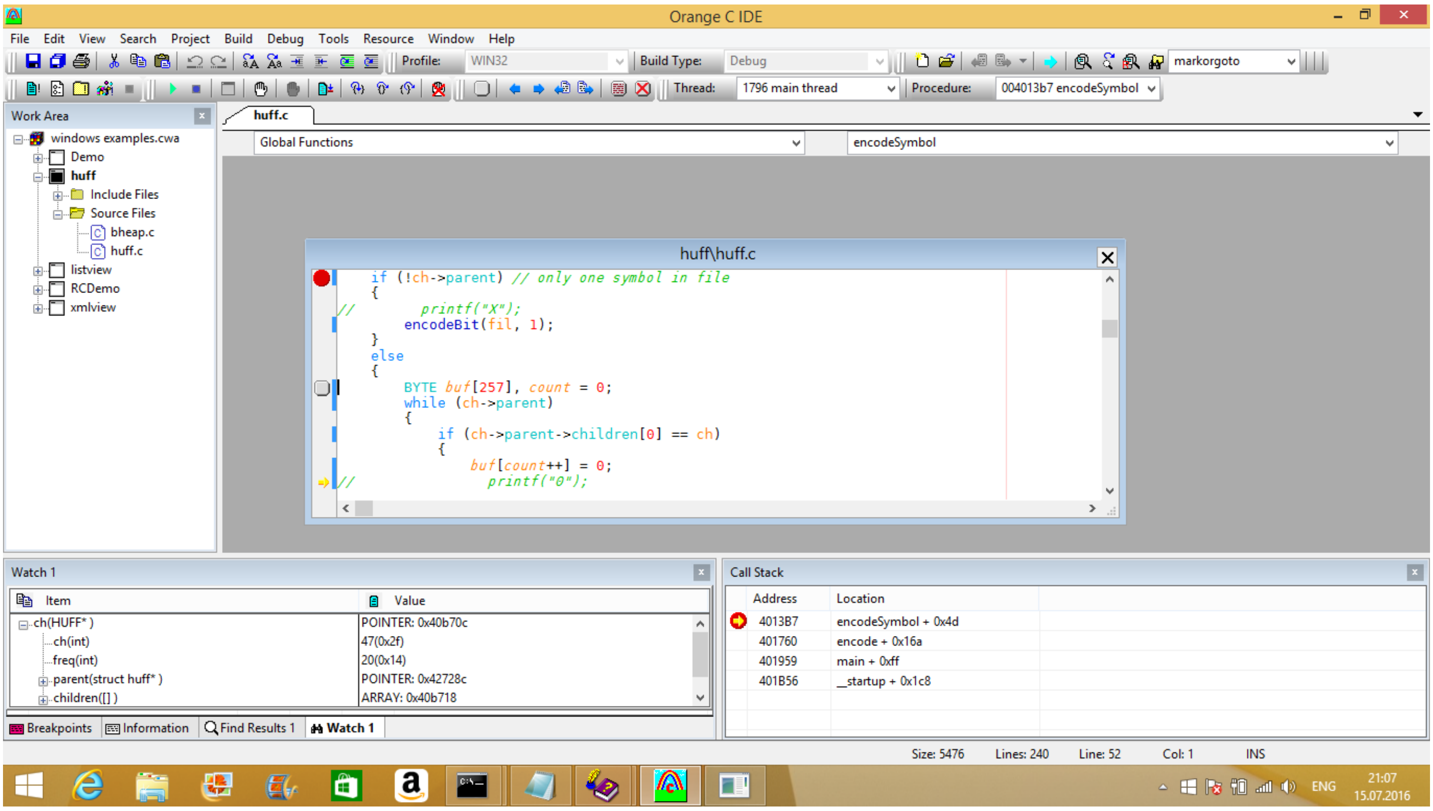
## Integrated Debugger

The debugger is started by using **Start Debugging** on the Debug Menu.

The debugger is a full-featured debugger.  Providing that the executable is compiled to enable debug information, all features are available once the debugger has started.  Note that breakpoints may be set while the debugger is stopped; this allows the possibility of setting breakpoints before the program starts.

When the debugger is started the program will start, and run until it hits a break point.  Alternately, press F5 or F10.  If F5 is pressed, the program will start running and run until it either ends or hits a breakpoint.  If F10 is pressed the program will start running and stop at the beginning of *main()* or *WinMain()*.

Following is an example of an editor window when the program is stopped.



```
    if (!ch->parent) // only one symbol in file
    {
//        printf("X");
        encodeBit(fil, 1);
    }
    else
    {
        BYTE buf[257], count = 0;
        while (ch->parent)
        {
            if (ch->parent->children[0] == ch)
            {
                buf[count++] = 0;
//                printf("0");
```

Watch 1

| Item | Value |
|---|---|
| ch(HUFF* ) | POINTER: 0x40b70c |
| ch(int) | 47(0x2f) |
| freq(int) | 20(0x14) |
| parent(struct huff* ) | POINTER: 0x42728c |
| children([] ) | ARRAY: 0x40b718 |

Call Stack

| Address | Location |
|---|---|
| 4013B7 | encodeSymbol + 0x4d |
| 401760 | encode + 0x16a |
| 401959 | main + 0xff |
| 401B56 | __startup + 0x1c8 |

The dark grey lines at the left indicate which lines of the program generated executable code. The red circle is a Break point. The yellow arrow indicates at what point the program has stopped.

Note that when the debugger stops on a function declaration, the processor stack is not set up yet. Therefore, it is not possible to accurately determine where the function's variables are. Local variables will appear as 'out of scope'. Single stepping past the function prototype will cause the processor stack to be initialized.

There are several menu items on the Editor Context Menu that relate to debugging. There is also a Debug Toolbar with some commonly used debugging functions such as starting the program, and single stepping. An additional button on the Debug Toolbar allows the possibility of stepping out of a function.

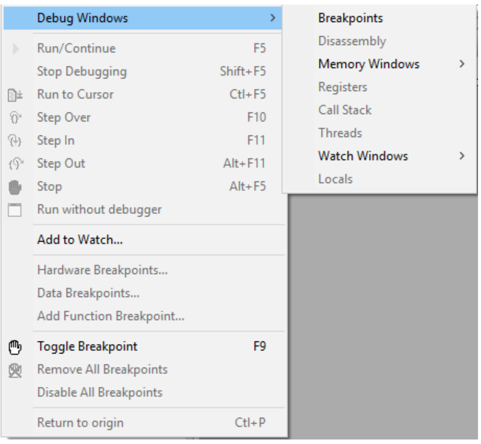
Single stepping can also be performed by pressing F10 for step over, and F11 for step into. F9 toggles breakpoints. F5 starts the program running when it is stopped.

The Project Debug Properties holds settings used for setting up debugging on a project by project basis. the Hints properties page has settings for enabling and disabling debug hints on a global basis.

Only one project can be started at a time, however if some projects are DLLs which are called by the active project, the debugger will allow symbolic debugging of them.

## Debug Menu

The Debug Menu has menu items related to debugging the project. It appears similar to what follows:



**Run/Continue** starts the debugger if it isn't running, at the active project. Debugging Windows are opened at this point. The program will stop at the first break point it encounters, or at the start of the *main()* or *WinMain()* function if no breakpoints exist. If the debugger is stopped in the bugger program execution is resumed.

**Stop Debugging** stops the debugger and closes Debugging Windows.

**Run To Cursor** runs the program. It will stop at the next break point encountered, when the program exits, or when the program flow reaches the cursor position in the current active edit window.

**Step Over** steps over the current statement. If the current statement is a function call the entire function is run; breakpoints in the function may still be hit though.

**Step In** steps into a function, if the current statement has one. Otherwise it steps over the statement.

**Step Out** runs until the current function exits, or until a breakpoint is hit.

**Stop** stops the program if it is running.

**Run Without Debugger** starts the program as a separate executable, without debugging. If it is a console program, a separate console window will be created for it to display data in.

**Add To Watch** opens the Add To Watch Dialog. This allows an expression to be added to the watch window.

**Hardware Breakpoints** opens the Hardware Breakpoints dialog.

**Data Breakpoints** opens the Data Breakpoint dialog to allow adding a data breakpoint on the word under the cursor.

**Add Function Breakpoint** prompts for a function name, then sets a breakpoint there.'

**Toggle Breakpoint** toggles the break point setting for the current line.  See Breakpoints for further information about breakpoints.

**Remove All Breakpoints** removes all breakpoints from the break point list.

**Disable All Breakpoints** disables all active breakpoints, but leaves the lines marked as breakpoints.

**Return To Origin** opens the window the current breakpoint is associated with and navigates to the breakpoint line.

The following menu exist on the **Debug Windows** submenu.  These items allow selection of various debug related windows, they are grayed out until the Integrated Debugger is started.

**Breakpoints** opens or closes the Breakpoints Window.   When the window is displayed a check mark will appear next to the menu item.

**Disassembly** opens or closes the Assembly Window.  When the window is displayed a check mark will appear next to the menu item.

**Memory** opens or closes one of four Memory Windows.  When the window is displayed a check mark will appear next to the menu item.

**Register** opens or closes the Register Window.  When the window is displayed a check mark will appear next to the menu item.

**Call Stack** opens or closes the Call Stack Window.  When the window is displayed a check mark will appear next to the menu item.

**Thread** opens or closes the Thread Window.  When the window is displayed a check mark will appear next to the menu item.

**Watch** opens or closes one of four Watch Windows.  When the window is displayed a check mark will appear next to the menu item.

**Locals** opens or closes the ~~Locals Window~~{linkID=655}.   When the window is displayed a check mark will appear next to the menu item.

## Debugging Windows

The IDE has several special windows used for helping to debug.  These windows are available from the Debug Windows Menu when the debugger is running.

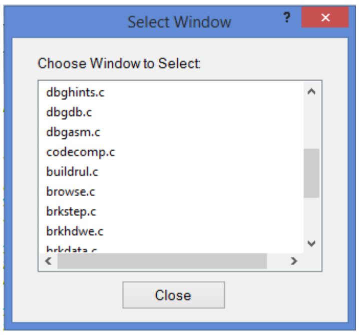
When the debugger is stopped, the IDE will close all debug windows.  However, it will remember which ones were open  and their position, and reopen them the next time the debugger is started.

The debugging windows are as follows:

Disassembly Window Shows the assembly language associated with the program
Breakpoints Window Shows all current breakpoints
Memory Window Shows a memory dump
Register Window Shows registers
Call Stack Window Shows a back trace of the processor stack
Threads Window Shows a list of threads
Watch Window Shows the value of variables
Locals Window Shows the values of local(auto) variables that are currently in scope

## Select Window Dialog

The Select Window Dialog is accessible from the *More Windows...* item of the Window Menu when there are enough edit windows open in the client area that the list of open windows will not fit on the Window Menu.  It is also available from the *More Windows...* item of the Client Context Menu. It shows a list of open windows and appears similar to the following:



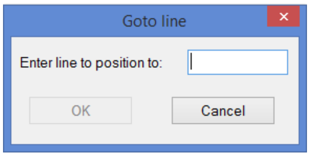
dbghints.c
dbgdb.c
dbgasm.c
codecomp.c
buildrul.c
browse.c
brkstep.c
brkhdwe.c
brkdata.c

Double Clicking on the name of a file name will cause that window to come to the top of the view area.

Special Windows such as the Watch Window will also appear in the client area if undocked, and will show up in this list.

## Goto Line Dialog

The Goto Line Dialog can be reached from the Navigation Toolbar or from the **Goto Line** option of the View Menu.  It allows a line number to be entered.  It appears similar to what follows:



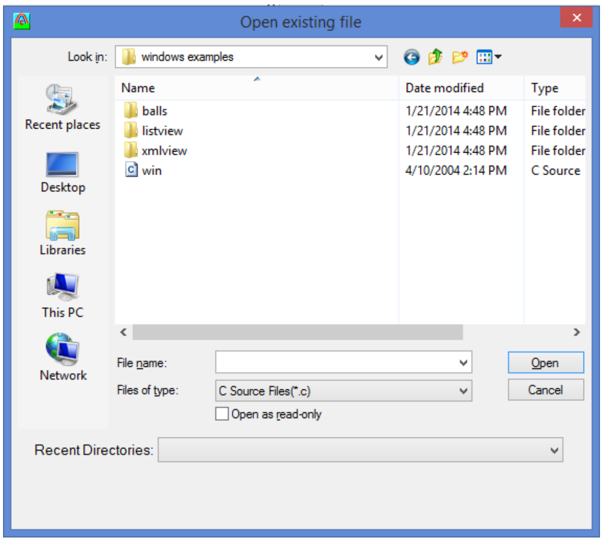
When **Ok** is clicked, the currently active editor window will be positioned to the specified line.

Note that it is possible to have line numbers displayed to the left of each line by selecting the **Show Line Numbers** of the Basic Editor Settings property page.

## Open File Dialog

The Open File Dialog is a standard windows dialog used for selecting files.  It is used to open files to edit, open workareas, locate projects to add to the workarea, and so forth.  In some cases such as opening files or adding them to a project, the open file dialog will accept a selection of multiple files.

The Open File Dialog appears as follows



Look in: windows examples

| Name | Date modified | Type |
|---|---|---|
| balls | 1/21/2014 4:48 PM | File folder |
| listview | 1/21/2014 4:48 PM | File folder |
| xmlview | 1/21/2014 4:48 PM | File folder |
| win | 4/10/2004 2:14 PM | C Source |

File name:

Files of type: C Source Files(*.c)

□ Open as read-only

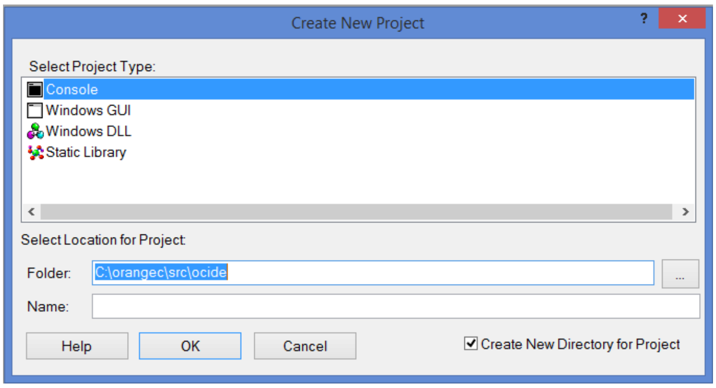
Recent Directories:

The enhanced Recent Directories combo box holds a list of directories that have been visited recently. It may be used as a convenient method for navigating to recently used directories.

The title bar changes to reflect the operation being performed. In this example, one or more text files may be opened.

## New Project Dialog

The New Project Dialog may be reached from the Workarea Context Menu, or from the **WorkArea->Add New Project** item of the File Menu. It allows selection of the name and location of a new project, and its type. The project type controls what kind of output file will be created, for example an executable file or a library. It appears similar to what follows:



Select Project Type:

- Console
- Windows GUI
- Windows DLL
- Static Library

Select Location for Project

Folder: C:\orangec\src\ocide

Name:

☑ Create New Directory for Project

To create a new project, first select its type. The following types are available:

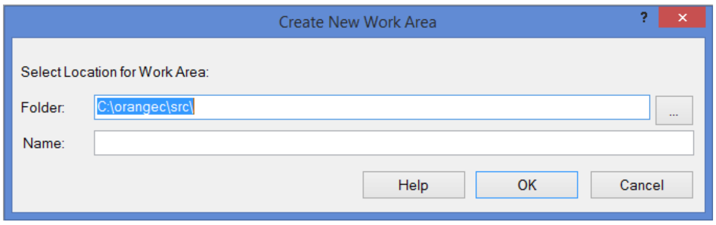
**Console** a console application, the type of command line program standard C runs as. Can also include windowing functions.
**Windows GUI** a program with a windows GUI
**Windows DLL** shared code and data in an executable library format of the type that Windows 32 bit requires
**Static Library** shared code and data in a library format that can be combined with other code when creating programs.
This is the type of library often found in non-windows environments.

Once the type has been chosen, select a name and location for the project. This name does not need to have an operating system extension; an extension will be added automatically based on the file type. This IDE will not automatically create the location, so it is necessary to select an existing directory. Pressing the button to the right of the location will bring up a dialog that allows searching for a directory.

Selecting **Create New Directory For Project** will result in the IDE creating a subdirectory with the projects name to put the project files in.

## New Work Area Dialog

The New Work Area Dialog is accessible by selecting **WorkArea->New** from the File Menu. It allows selection of the name and location of a new workarea. It appears similar to what follows:



Select Location for Work Area:

Folder: C:\orangec\src\

Name:

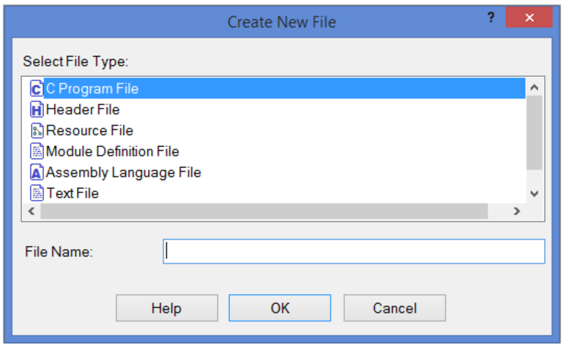
In this dialog, select a name and location for the project. This name does not need to have an operating system extension; an extension will be added automatically based on the file type. This IDE will not automatically create the location, so it is necessary to select an existing directory. Pressing the button to the right of the location will bring up a dialog that allows searching for a directory.

When typing the file name, the IDE will automatically attempt to update the folder name by adding a subdirectory name based on the file name. However, this can be edited after the filename is typed in to put the workarea in an arbitrary folder.

## New File Dialog

The New File Dialog is accessible from the Project Context Menu, and by selecting **File->New** from the File Menu. allows selection of the name and location of a new file, and its type. The file type controls what extension the file will get, and thus how the file will be processed. For example C language files will be compiled with the C compiler.

The file type may also cause the file to be automatically sorted into a default folder, The New File Dialog appears similar to what follows:



Create New File

Select File Type:

- C Program File
- Header File
- Resource File
- Module Definition File
- Assembly Language File
- Text File

File Name:

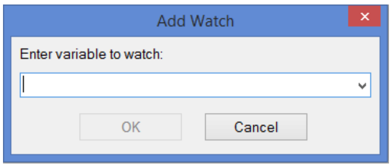
To create a new file, first select its type.  The following types are available:

**C Program File** Program source code(.C) to be compiled with the compiler compiled with OCC
**Header File** Header Information(.H) for program source code included in other files
**Resource File** Window Resource (.RC) file compiled with ORC
**Module Definition File** Module Definition (.DEF) file sometimes used in creating windows programs
**Assembly Language File** Assembly Language source code (.asm) assembled with OASM
**Text File** Plain Text (.txt) that isn't compiled
This is the type of library often found in non-windows environments.

Once the type has been chosen, select a name for the file.  This name does not need to have an operating system extension; an extension will be added automatically based on the file type.  The IDE will place the new file in the project when it is first saved.

## Add To Watch Dialog

The Add To Watch Dialog is accessible from the Editor Context Menu, and from the Watch Window Context Menu.  It allows entry of the name of a variable.  It appears similar to what follows:



Add Watch

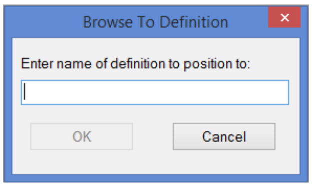
Enter variable to watch:

Any expression may be entered into the text field.  The combo box remembers recently selected expressions.

When **Ok** is pressed, the variable will appear in the Watch Window along with its value.

## Browse To Dialog

The Browse To Dialog is accessible from the View->Browse Menu.  It allows entry of a variable or function name to locate.  It appears similar to what follows:



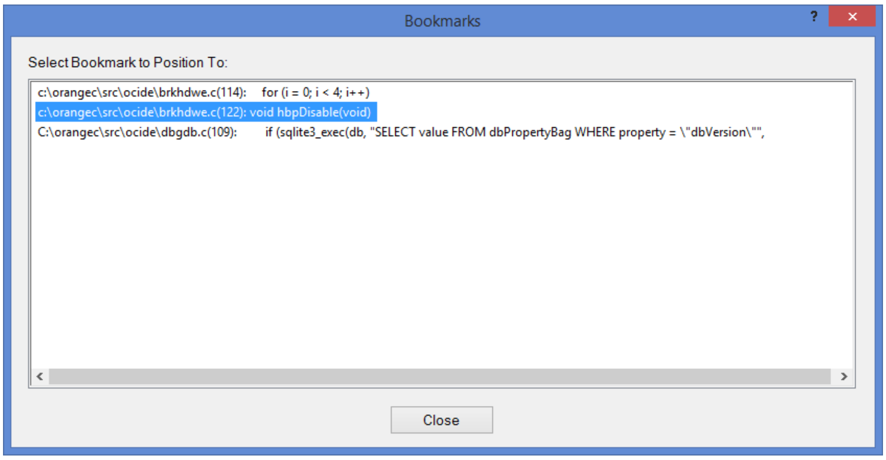
Browse To Definition

Enter name of definition to position to:

After pressing Ok, the definition of the name is located and shown in an Editor Window.

## Bookmark Dialog

The Bookmark Dialog is accessed from the View->Bookmarks Menu.  It offers a way to select from a list of bookmarks.  It appears similar to what follows:



Bookmarks

Select Bookmark to Position To:

```
c:\orangec\src\ocide\brkhdwe.c(114):    for (i = 0; i < 4; i++)
c:\orangec\src\ocide\brkhdwe.c(122): void hbpDisable(void)
C:\orangec\src\ocide\dbgdb.c(109):        if (sqlite3_exec(db, "SELECT value FROM dbPropertyBag WHERE property = \"dbVersion\"",
```

Double-Clicking on a bookmark opens an Editor Window which shows the line the bookmark is on.

## Bookmarks

Bookmarks are used to mark positions in the text for easy traversal. They may be turned on and off for any line in any files. A visual indication is given to the left of the editor window as to which lines have bookmarks enabled.

The View Bookmarks Menu has various items which can be used to turn bookmarks on and off, and traverse through the list of bookmarks. The Bookmark Toolbar has the same items. Bookmarks may also be toggled via the Editor Context Menu The **Show Bookmarks** item located in some of these locations brings up the Bookmark Dialog, which can be used to select an arbitrary bookmark from the list.

## Breakpoints

Breakpoints are set in order to bring the program to a stop. This is useful because variables cannot be analyzed while the program is running. Typical things to do when a break point is triggered are to look at variables in the Watch Window or by moving the mouse over the variable name, analyze the call stack back trace, or look at the list of running threads.

Breakpoints can be set in a variety of ways, for example from the Debug Toolbar, from the Debug Menu,
from the Editor Context Menu, or by selecting a line in an editor window and pressing F9.

Breakpoints may be set while the Integrated Debugger is stopped, but only take effect when the debugger is running.

Breakpoints may also be set while in the Disassembly Window. In this case a breakpoint can be set on arbitrary machine instructions, not just at the beginning of a line.

A Breakpoint Window can show a list of current breakpoints and allow individually enabling or disabling them.

When no breakpoints are set, the program stops at the *main()* or *Winmain()* function when run, as if a breakpoint had been set there.

Another type of break point is implicitly set when the **Run To** item on the Debug Toolbar is used. A break point is set at the cursor address to force the program to stop there.

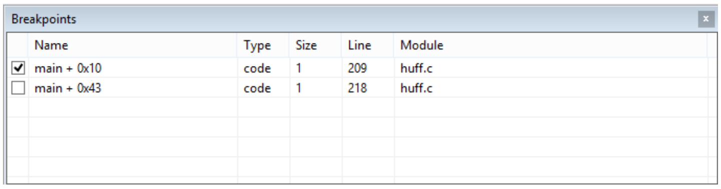
A breakpoint may also be set with the mouse when the debugger is running. Simply double click in the area on the left of the Editor Window.

The debugger supports Hardware Breakpoints. This type of break point is supported by the microprocessor itself, and is used to track reads and writes to variables. It is a simpler but faster form of tracking changes to variables, useful for small quantities such as single characters or integers.

Finally, the debugger supports Data Breakpoints. This type of breakpoint detects when a variable changes. It can handle more complex variables than the corresponding Hardware Breakpoints, such as entire structures or arrays. However, it uses a form of detection that has the potential to slow the program down somewhat while detection of changes is being done.

## Breakpoints Window

The Breakpoints window allows viewing and editing a list of breakpoints. It looks similar to what follows:



Breakpoints

| | Name | Type | Size | Line | Module |
|---|---|---|---|---|---|
| ☑ | main + 0x10 | code | 1 | 209 | huff.c |
| □ | main + 0x43 | code | 1 | 218 | huff.c |

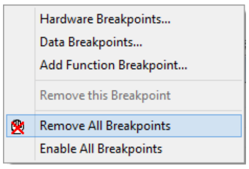
This example has two breakpoints; one will stop the program and the other is currently disabled. This window also shows data and hardware breakpoints, if they have been selected.

Right clicking in this window brings up the Breakpoint Context Menu.

Clicking on a code breakpoint takes one to the line the breakpoint is defined on. Clicking on data or hardware breakpoints has no effect.

## Breakpoint Context Menu

The breakpoint context menu has several items that may be used to modify the contents of the Breakpoints Window.



**Hardware Breakpoints** allows creating a Hardware Breakpoint

**Data Breakpoints** allows creating a Data Breakpoint

**Add Function Breakpoint** allows creating a breakpoint at the beginning of a function

**Remove This Breakpoint** removes the breakpoint that was right-clicked on from the list of breakpoints

**Remove All Breakpoints** remove all breakpoints removes all breakpoints from the list

**Enable All Breakpoints** enables or disables all breakpoints in the list

## Integrated Platform Help

The IDE has four help files. This is the first one, which is accessed by pressing F1 or by using the help buttons arranged on various dialogs. The second one is the Tools Help File, which gives specific information about the command line tools the IDE utilizes to create programs. The third one is the RTL Help File, which gives specific information about run-time functions available in the C language. The fourth one is the Language Help File

These help files are all accessible from the Help Menu.

Run Time Library help is also available for the word under the cursor by pressing SHIFT-F1.

## Browse Information

The IDE offers source code browsing. Placing a cursor on a variable or structure name, the selecting **Browse to Definiton or Browse to Declaration** on the View Browse Menu will browse to the variable or structure declaration. Alternatively the Browse To selection will open the Browse To Dialog and allow a name to be entered directly.

The browse function requires compiler support and must be enabled from the General Properties Dialog. After enabling it, the project must be recompiled with the Build All option. In C++ code it is also possible to browse to a function which has been referenced through a structure instance, but this requires that Code Completion also be enabled.

Various C++ language constructructs are supported, for example the namespace selector.

Browsing is only possible within C language files and headers.  Note that the browse information file (which is handled in the background) may become very large for large programs.

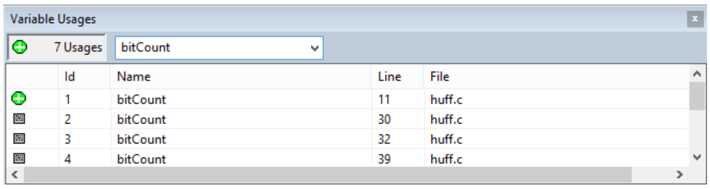
Browse information is also used by the Jump List and the Variable Usages Window.

Browsing is enabled in the Hints Configuration.

If a browse is requested, and browsing is not enabled, the IDE will ask if the workarea should be rebuilt with browsing enabled.  If the answer is yes, the workarea will be rebuilt as if by the **Rebuild All** selection on the Build Menu, and then the browse will complete.

## Variable Usages Window

The Variable Usages window is used to view the position of the definition and usages of a variable.   It looks similar to what follows:



Variable Usages

7 Usages  bitCount

| | Id | Name | Line | File |
|---|---|---|---|---|
| ⊕ | 1 | bitCount | 11 | huff.c |
| ▣ | 2 | bitCount | 30 | huff.c |
| ▣ | 3 | bitCount | 32 | huff.c |
| ▣ | 4 | bitCount | 39 | huff.c |

On this window, type the name of the variable in the combo box.   The browse information will be used to determine where the variable is defined (shown with the green plus) and where it is used (shown with the gray box).   It is also possible to see the location where the variable is declared, which will be indicated by a yellow triangle with an exclamation point.

Clicking on a row takes one to the line of source code with the definition or usage.

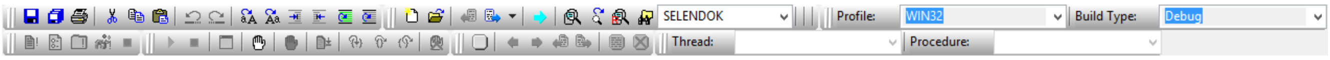
## Printing

The IDE provides a range of standard printing features, including multiple copy print and collating.  The contents of any open Editor Window may be directed to the printer by using the **Print** item on the File Menu.

This documentation will not attempt to cover the features of the windows printer dialog.  However, the topic Printer Settings  discusses configuration of printouts.

## Toolbars

Orange C IDE has six toolbars.  In general the items on the toolbar also appear somewhere in the main menu or in context menus.  The toolbars appear this way when the IDE is distributed:



SELENDOK

Profile: WIN32

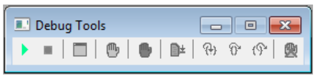
Build Type: Debug

Thread:

Procedure:

The toolbars are customizable via the Toolbar Customization dialog.  Additionally, if the ALT key is held down the mouse can be used to drag buttons around on a toolbar, or off the toolbar to remove them.

The toolbars may be dragged to be in a different order by holding the grip bar at the left.  They may also be dragged off the menu bar entirely and placed in the workarea.  A toolbar in the workarea looks like this:

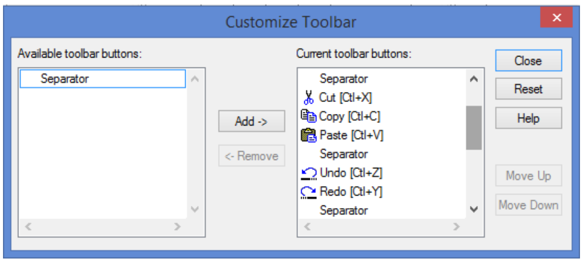


For more information consult the following topics:

Edit Toolbar
Navigation Toolbar
Bookmark Toolbar
Build Toolbar
Debug Toolbar
Thread Toolbar
Build Type Toolbar

## Toolbar Customization Dialog

When a toolbar is being customized, the Toolbar Customization Dialog appears.  This is a standard windows dialog box.  It appears similar to what follows:



Customize Toolbar

Available toolbar buttons:

Separator

Current toolbar buttons:

Separator
Cut [Ctl+X]
Copy [Ctl+C]
Paste [Ctl+V]
Separator
Undo [Ctl+Z]
Redo [Ctl+Y]
Separator

This dialog configures placement of items in the toolbar, but does not affect where on the display the toolbar is placed.

The left pane shows items that aren't in the toolbar.  The right pane shows items currently in the toolbar, in the order which they appear.

To move an item into the toolbar, select the item in the left pane and its position in the right pane, and press the **Add** button.

To remove an item from the toolbar, select it in the right pane and press the **Remove** button.

Separators may be inserted anywhere in the toolbar.

**Close** closes the dialog box; changes are accepted.

**Reset** resets the toolbar to its default configuration.

**Move up** moves an item upward in the box, which is to the left in the toolbar.

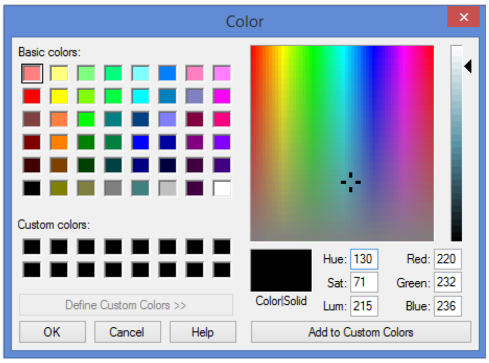
**Move down** moves an item downward in the box, which is to the right in the toolbar.

**Help** displays this text.

In addition to customizing the toolbar with this Dialog, pressing shift and clicking on an item in an active toolbar will allow you to drag the item elsewhere on the tool bar, or drag it off the toolbar

## Color Dialog

The Color Dialog is accessed either from the General Properties Dialog (to set colorization settings) or from the Resource Image Window. It appears similar to what follows:

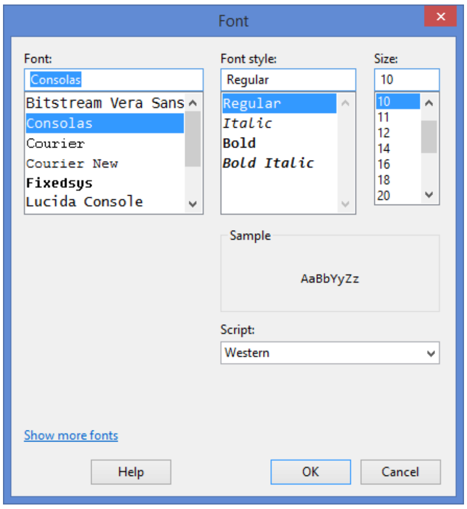


Hue: 130
Sat: 71
Lum: 215
Red: 220
Green: 232
Blue: 236

The dialog is shown here in its expanded state; normally when the dialog is first opened the palette at the right is not visible. To open it use the **Define Custom Colors** button. To select a color choose from the palette on the left. Alternatively add a color from the palette on the right to the Custom Colors box on the left, then choose the color from the custom colors box.

## Font Dialog

The Font Dialog is accessed from the General Properties Dialog (to set the font for Editor Windows) or for a Resource. It appears similar to what follows:

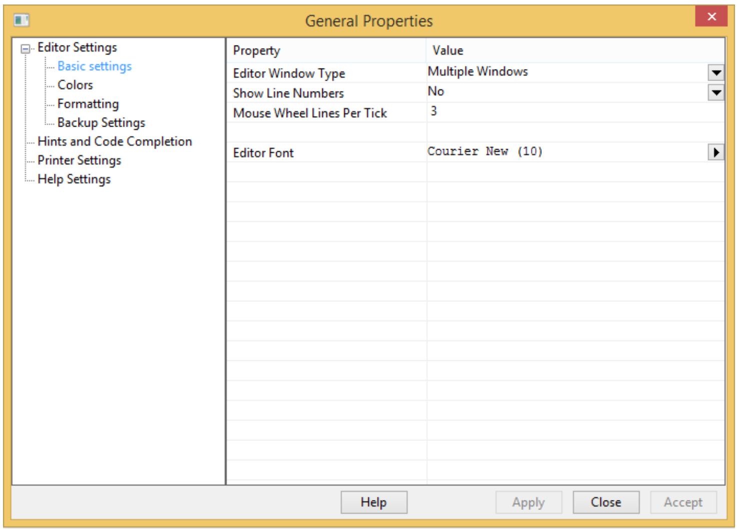


Font: Consolas
Font style: Regular
Size: 10
Script: Western

The **Font**, **Style** and **Size** items set font parameters. The IDE uses bold and italicized fonts implicitly for colorization, so in general the style should be left as normal.

## Basic Settings

The Basic Settings properties page is inside the General Properties Dialog. It allows modification of several of the most basic editor settings. It appears similar to what follows:



| Property | Value |
|---|---|
| Editor Window Type | Multiple Windows |
| Show Line Numbers | No |
| Mouse Wheel Lines Per Tick | 3 |
| Editor Font | Courier New (10) |

The following properties are available on this page:

**Editor Window Type** sets the way edit windows are displayed. It may be set to one of the following values:
Multiple Windows multiple overlapping windows will be displayed. Clicking on one's title bar will bring it to the front
Tabbed Windows only one window will be displayed and it will take up the available area.

**Show Line Numbers** if set to yes, line numbers will be displayed at the left of each editor window. Otherwise they will not be displayed. The current line number is also always shown in the status bar.

**Mouse Wheel Lines Per Tick** selects the number of lines to scroll up or down if the mouse wheel is moved one click.

**Editor Font** selects the font to be used in the editor. When the arrow to the right of the font name is clicked, a Font Dialog will appear to help locate a new font.

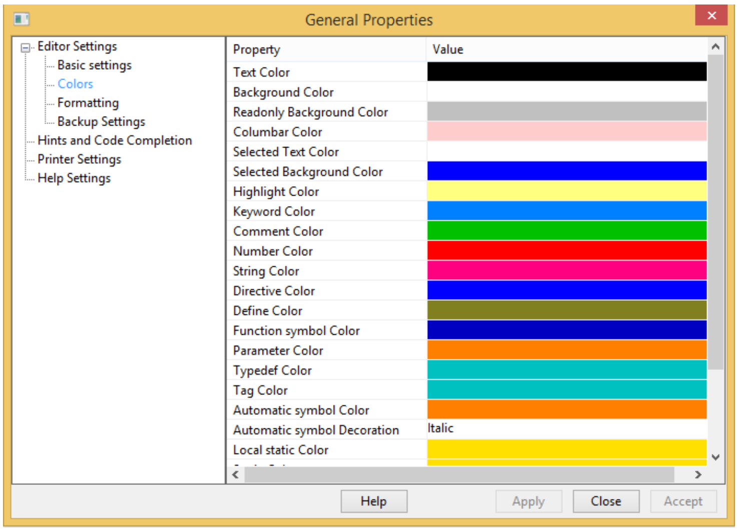
**Apply** saves changes but leaves the dialog open

**Close** closes the dialog without saving changes

**Accept** saves changes then closes the dialog

# Colors

The Color Settings properties page is inside the General Properties Dialog. It allows modification of colors used in the editor presentation. It appears similar to what follows:



| Property | Value |
|---|---|
| Text Color | ⬛ |
| Background Color | ⬜ |
| Readonly Background Color | ⬜ |
| Columbar Color | ⬜ |
| Selected Text Color | ⬜ |
| Selected Background Color | 🟦 |
| Highlight Color | 🟨 |
| Keyword Color | 🟦 |
| Comment Color | 🟩 |
| Number Color | 🟥 |
| String Color | 🟪 |
| Directive Color | 🟦 |
| Define Color | 🟫 |
| Function symbol Color | 🟦 |
| Parameter Color | 🟧 |
| Typedef Color | 🟦 |
| Tag Color | 🟦 |
| Automatic symbol Color | 🟧 |
| Automatic symbol Decoration | Italic |
| Local static Color | 🟨 |

This page has a list of available colors, and shows a bar with each color next to its name. Clicking on the arrow at the right of the bar brings up a Color Dialog which may be used to edit the color.

Note that the auto-coloring may be turned off via the ~~Colorization~~ item in the Formatting Settings page.

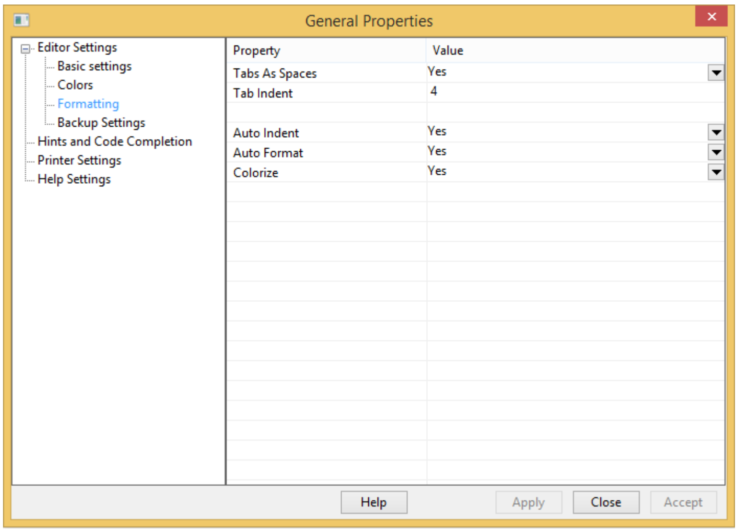
**Apply** saves changes but leaves the dialog open

**Close** closes the dialog without saving changes

**Accept** saves changes then closes the dialog

# Formatting

The Formatting Settings properties page is inside the General Properties Dialog. It allows modification of editor settings related to formatting the text. It appears similar to what follows:



| Property | Value |
|---|---|
| Tabs As Spaces | Yes |
| Tab Indent | 4 |
| | |
| Auto Indent | Yes |
| Auto Format | Yes |
| Colorize | Yes |

The following properties are available on this page:

**Tabs As Spaces** when set to Yes, pressing the tab key will result in a number of spaces being embedded into the text, sufficient to take the cursor to the next tab stop. When set to no, pressing the tab key will result in a single special character meaning tab being embedded in the text. The IDE will interpret this character to mean move the cursor to the next tab stop.

**Tab Indent** this is the number of single-character column positions between tab stops.

**Auto Indent** when set to yes, pressing **Enter** will cause the IDE to insert enough spaces or tabs to bring the cursor to the logical position text on the line should appear at. For most lines this means the cursor will be lined up with the position of the first character of the previous line. However for some C language statements such as if statements there is an additional indentation to the next tab stop, to make it easier to remember to indent such lines.

**Auto Format** when set to yes some formatting will be performed automatically. This includes moving preprocessor directives to the beginning of the line, and lining up close-brackets in the same column with the corresponding open-bracket.

**Colorize** when set to yes the colors on the Color Settings page will be applied to the text in Edit Windows, otherwise all text will be black.

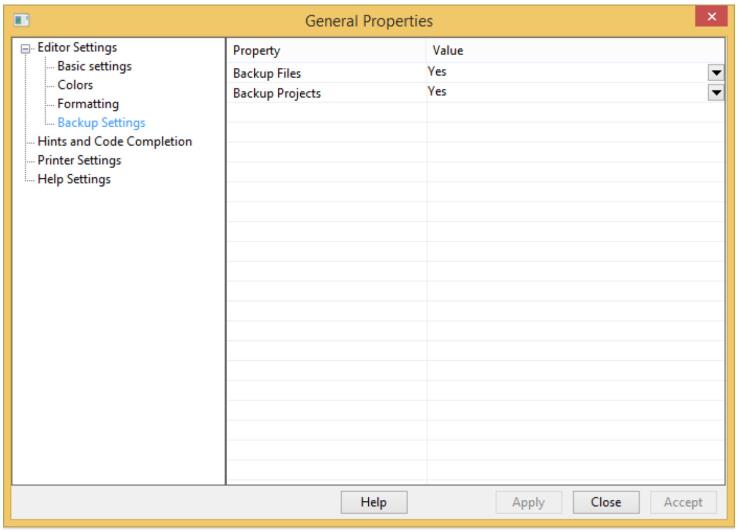
**Apply** saves changes but leaves the dialog open

**Close** closes the dialog without saving changes

**Accept** saves changes then closes the dialog

## Backup Settings

The Backup Settings is inside the General Properties Dialog. It allows modification of auto-backup settings. It appears similar to what follows:



| Property | Value |
|---|---|
| Backup Files | Yes |
| Backup Projects | Yes |

The backup settings, when set to yes, cause the corresponding types of files to be backed up just prior to saving a new version to disk. To create a name for the backup file, the original file will have the string '.bak' appended to it.

For example, to back up a file **MySourceFile.c**, the IDE will create a new file **MySourceFile.c.bak.**

The following properties are available on this page:

**Backup Files** if set to yes, source code files will be backed up before being changed

**Backup Projects** if set to yes, projects and workareas will be backed up before being changed

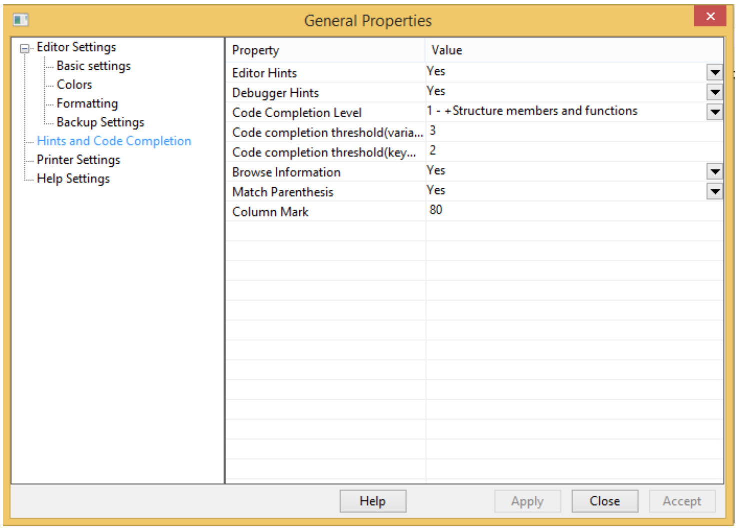
**Apply** saves changes but leaves the dialog open

**Close** closes the dialog without saving changes

**Accept** saves changes then closes the dialog

## Hints And Code Completion

The Hints And Code Completion Settings is inside the General Properties Dialog. It allows modification of editor settings related to hints and code completion. It appears similar to what follows:



| Property | Value |
|---|---|
| Editor Hints | Yes |
| Debugger Hints | Yes |
| Code Completion Level | 1 - +Structure members and functions |
| Code completion threshold(varia... | 3 |
| Code completion threshold(key... | 2 |
| Browse Information | Yes |
| Match Parenthesis | Yes |
| Column Mark | 80 |

The following properties are available on this page:

**Editor Hints** when set to yes, editor hints will be shown while not debugging. These involve showing the declaration of types and variables.

**Debugger Hints** when set to yes, debugger hints will be shown while debugging. These involve showing the value of variables. At this time only simple types are displayed, structures and arrays are not displayed in the hint.

**Code Completion Level** sets the amount of code completion help that is desirable. Code Completion windows give help for accessing keywords, functions, structures, and variables but the constant popups can be an annoyance. There are three code completion levels:

Level 0 Code completion is off
Level 1 Accessing structure members will result in a small popup that allows selection of a member; typing in something that looks like a function call will result in a hint
Level 2 The level 1 functionality, plus a small popup that allows selection of variables and keywords

Note that code completion only works for entities that have been defined; for example hints for strcmp may not be available if the appropriate header file which prototypes strcmp is not included as a #include in the source code.

**Code Completion Threshold(Variables)** sets the minimum number of characters that have to be typed as part of a variable name before the level 2 popup will appear.

**Code Completion Threshold(Keywords)** sets the minimum number of characters that have to be typed as part of a variable name before the level 2 popup will appear.

**Browse Information** when set to yes, compiling the project will generate information that can aid in locating the definition of a function. At this point, the Browse Menu items will allow locating the definition of a variable or function name.

**Match Parenthesis** when set to yes, positioning the cursor next to a bracket or parenthesis will locate a matching bracket or parenthesis and highlight both. For example positioning the cursor on an open parenthesis will locate the matching close parenthesis and highlight both.

**Column Mark** allows setting of a column position, at which a vertical line will be drawn. The vertical line gives a visual aid as to how long the line is. For example it could be used to try to limit how long each line is (by placing line breaks in an appropriate position).

**Apply** saves changes but leaves the dialog open

**Close** closes the dialog without saving changes

**Accept** saves changes then closes the dialog

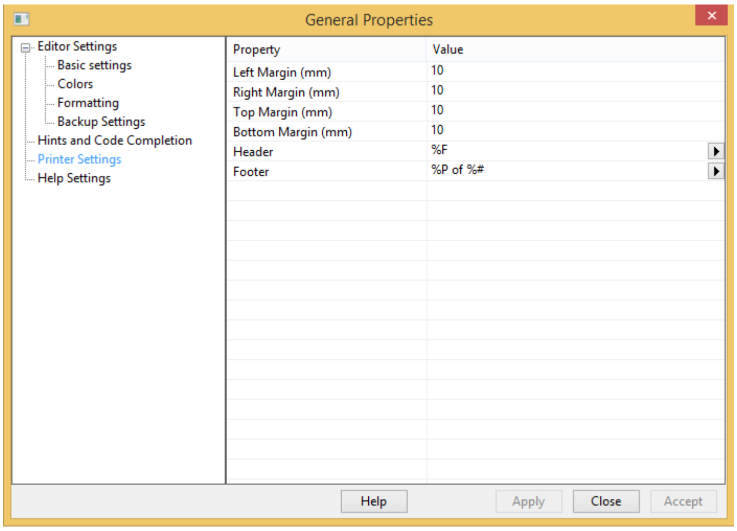
## Code Completion

Code Completion is a feature of the IDE that allows it to dynamically show a function prototype, or a list of structure members, while code is being typed in the editor. In the C language this feature is relatively straightforward. However in C++ there are a number of language features that are explicitly supported when typing a name for completion. Some of these are as follows:

Namespace specifiers may be used to select a namespace.
Using statements will usually be honored.
Multiple functions may be listed if there are several overload matches
If the function currently being edited is a member function, its parent class will be used to filter the options.

## Printer Settings

The Printer Settings properties page is inside the General Properties Dialog. It allows modification of printer-related settings. It appears similar to what follows:



| Property | Value |
|---|---|
| Left Margin (mm) | 10 |
| Right Margin (mm) | 10 |
| Top Margin (mm) | 10 |
| Bottom Margin (mm) | 10 |
| Header | %F |
| Footer | %P of %# |

The following properties are available on this page:

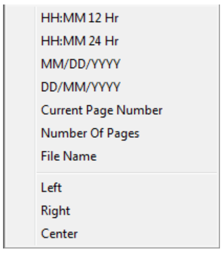
**Left Margin** is the size of the left margin, in millimeters.

**Right Margin** is the size of the right margin, in millimeters.

**Top Margin** is the size of the top margin, in millimeters.

**Bottom Margin** is the size of the bottom margin, in millimeters.

**Header** and **Footer** give settings for formatting that should be performed on optional header and footer lines. It isn't necessary to remember the shorthand for the options, as pressing the arrow to the right of the setting gives a menu with the options on it. The menu appears as follows:



This menu has the following options, which are displayed on the settings line as shown in parenthesis:

**Time Formats** include 12 hour**(%T)** and 24 hour**(%t)** time formats for the current time

**Date Formats** include the US date**(%D)** and European date**(%d)** formats for the current date

**Page Number** of the page currently being printed**(%P)**

**Number of Pages** in source file being printed**(%#)**

**File Name** of source file being printed**(%F)**

**Left(%L)**, **Right(%R)**, and **Center(%C)** allow positioning of the text in various places on the line

**Apply** saves changes but leaves the dialog open

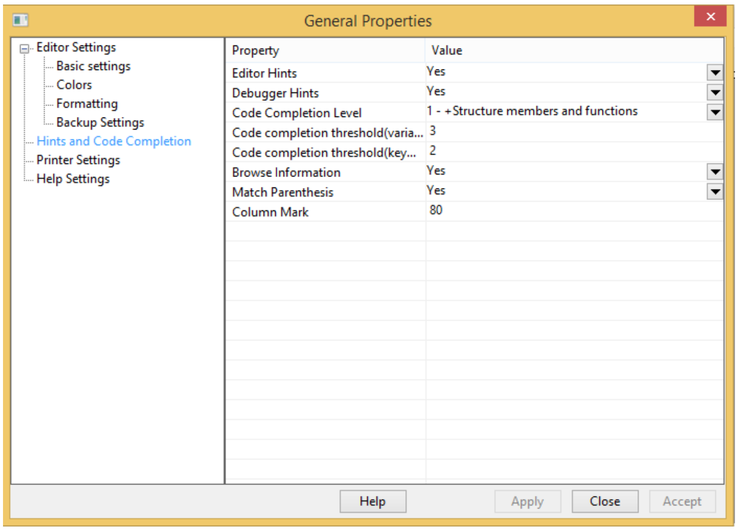
**Close** closes the dialog without saving changes

**Accept** saves changes then closes the dialog

## General Properties Dialog

The General Properties Dialog is accessible by selecting **Properties** on the Tool Menu. It is used for general IDE configuration which is unrelated to projects, such as color and font configuration.

On the left hand side of the dialog is a tree view that lists the available pages; on the right hand side properties for a page will be shown. Following is an example of how one of the general properties dialog pages appears.



| Property | Value |
|---|---|
| Editor Hints | Yes |
| Debugger Hints | Yes |
| Code Completion Level | 1 - +Structure members and functions |
| Code completion threshold(varia... | 3 |
| Code completion threshold(key... | 2 |
| Browse Information | Yes |
| Match Parenthesis | Yes |
| Column Mark | 80 |

This dialog has the following panes:

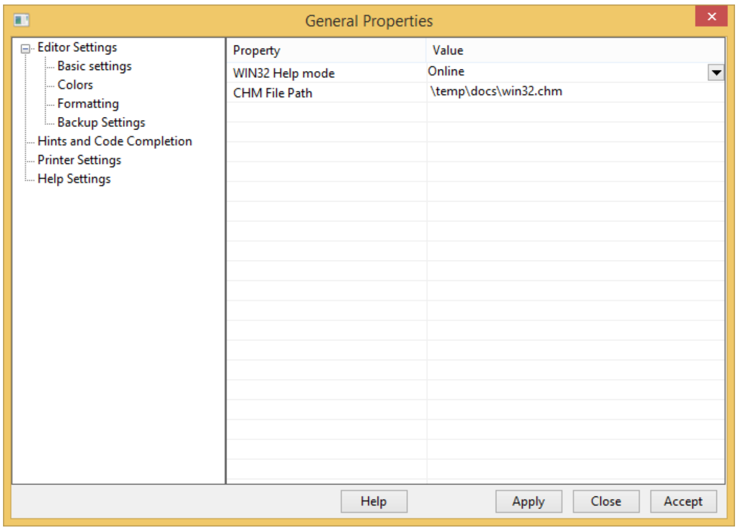
**Editor Settings**
Basic Settings Simple settings such as the font
Colors Colors to be used when highlighting source code elements
Formatting Settings related to text spacing and formatting
Backup Settings File backup parameters

**Other Settings**
Hints and Code Completion Settings related to automatic help given by the editor
Printer Settings Printer Settings
Win32 Help configuration Help file settings for the WIN32 help

Properties specific to building programs are on the Project Properties dialog

## WIN32 Help Configuration

The WIN32 Help Configuration properties page is inside the General Properties Dialog. It allows modification of settings related to setting up WIN32 help. It appears similar to what follows:



| Property | Value |
|---|---|
| WIN32 Help mode | Online |
| CHM File Path | \temp\docs\win32.chm |

When WIN32 help is configured the key sequence **CTL-F1** will use the WIN32 help to locate the word under the cursor.

The following properties are available on this page:

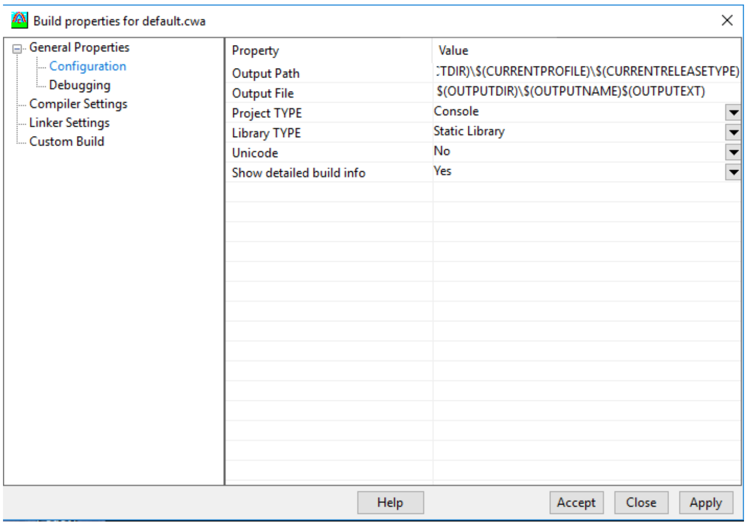
**Win32 Help Mode** is the type of win32 help that is configured.   It is one of the following:

**Online Help** This uses an internet connection to access MSDN in online format
**Windows 8 Offline help** This allows using the MSDN help on a local computer
  A configuration step downloads the help
**WIN32.CHM** This uses the old Win32 help, which has been converted to CHM format.

**CHM File Path** is the complete path where the WIN32.CHM help file resides.  It is only used if the mode is WIN32.CHM

## General Project Configuration Properties

The General Project Configuration Properties page is inside the Project Properties Dialog.  It allows modification of project settings related to the type of project.  It appears similar to what follows:



Build properties for default.cwa

| Property | Value |
|---|---|
| Output Path | :TDIR)\$(CURRENTPROFILE)\$(CURRENTRELEASETYPE) |
| Output File | $(OUTPUTDIR)\$(OUTPUTNAME)$(OUTPUTEXT) |
| Project TYPE | Console |
| Library TYPE | Static Library |
| Unicode | No |
| Show detailed build info | Yes |

The following properties are available on this page:

**Output Path** selects the place to store the intermediate compilation files, and the output executable or library file.  It is normally set to **$(PROJECTDIR)**.

**Output File** selects the name of the output executable or library.  It is normally set to **$(OUTPUTDIR)\$(OUTPUTNAME)$(OUTPUTEXT).** where **$(OUTPUTDIR)** is derived from the output path, **$(OUTPUTNAME)** is from the project name, and **$(OUTPUTEXT)** is an operating-system dependent file extension such as .EXE .DLL or .LIB that depends on the Project Type setting.

**Project Type** selects the project type.  The following project types are available:

**CONSOLE** an application designed to have a text mode interface.  This is the type of application C compilers normally create.
**GUI** an application designed to have a windowing interface.
**DLL** a shared library that holds code that may be executed by the operating system
**Static Library** a shared library that holds code that may be combined with other code to make an application or DLL.
**DOS** an MSDOS application.
**RAW** an application that is raw, designed for no OS in particular.  The runtime library would not work with it very well.

**Library Type** selects the type of C Language run-time library to use for standard functions like `printf`, when creating an application executable.  The following library types are available:

**None** don't use a library.  Project types other than Static Library and Raw that don't use a library will not run under an OS.
**Static Library** take the code from the library and put it directly in the application
**LSCRTL.DLL** use the LSCRTL.DLL that comes with this package as the library.
**CRTDLL.DLL** use the OS file CRTDLL.DLL as the library
**MSVCRT.DLL** use the OS file MSVCRT.DLL as the library

Note that in addition to the C language library, the IDE may add a static library that creates links to windowing functions.

**Unicode** when set to Yes, the UNICODE version of the windowing functions will be selected.  Otherwise, the ASCII version will be selected.  This does not affect functions like `printf` which are in the C language runtime library.

**Show detailed build info**, when set to YES, displays the commands used during the build process, as the project is being built.

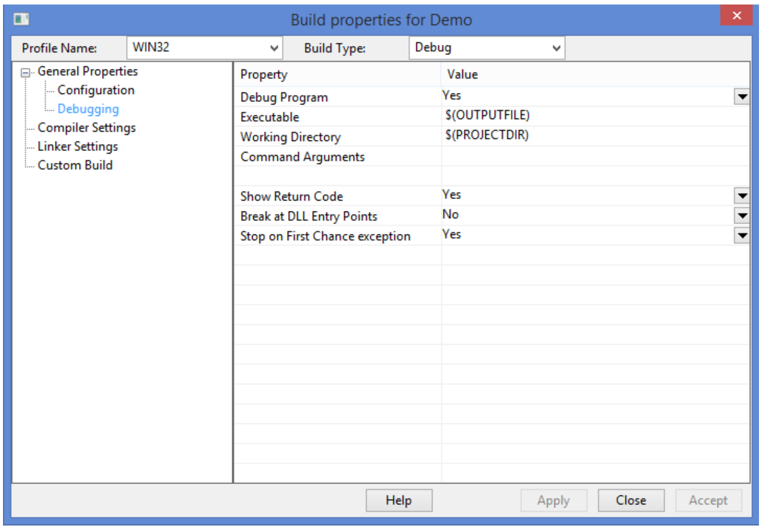
**Apply** saves changes but leaves the dialog open

**Close** closes the dialog without saving changes

**Accept** saves changes then closes the dialog

## Debug Project Config Properties

The Debug Project Configuration Properties page is inside the Project Properties Dialog.  It allows modification of project settings related to debugging the project. It appears similar to what follows:



Profile Name: WIN32 Build Type: Debug

| Property | Value |
|---|---|
| Debug Program | Yes |
| Executable | $(OUTPUTFILE) |
| Working Directory | $(PROJECTDIR) |
| Command Arguments | |
| | |
| Show Return Code | Yes |
| Break at DLL Entry Points | No |
| Stop on First Chance exception | Yes |

Note there is also a setting on the Hints And Code Completion properties page which enables or disables debugger hints. Debugger hints show the value of variables when the cursor is placed over them.

The following properties are available on this page:

**Create Debug Info**, when set to yes creates a debug info file when the project is built

**Executable** is the path to an executable to run to debug the program. Normally it is **$(OUTPUTFILE)** which means the program generated by the project. This is the same file specified by the **Output File** setting on the General Project Configuration Page. It may be necessary to change it to something else, for example when attempting to debug a DLL.

**Working Directory** the path that the executed program will retrieve when it uses functions like `getpwd` or `GetCurrentDirectory`.

**Command Arguments** any arguments that should be passed to the executed program on the command line

**Show Return Code** when set to yes, a box will appear after the program exits, showing the value that was passed to the operating system when it exited.

**Break At DLL Entry Points** when set to yes, the debugger will detect the entry points of DLLs that have debug information, and stop in their initialization function such as `DLLMain`.

**Stop On First Chance Exception** when set to no, the debugger will give the program a chance to handle any exception generated by the operating system, before declaring an error and stopping. When set to yes, the debugger will stop immediately when an exception is raised, without giving the program a chance to handle it.

**Apply** saves changes but leaves the dialog open

**Close** closes the dialog without saving changes

**Accept** saves changes then closes the dialog

# Project Properties Dialog

The Project Properties Dialog is accessible by selecting **Properties** on the Workarea Context Menu, the Project Context Menu, or the File Context Menu. It can also be reached by selecting **Properties** from the Project Menu; this specifically opens a Properties window associated with the active project. It is used for general IDE configuration related to building projects.

The Project Properties are tiered in several levels. The same settings appear internally on each level.

The top level is for the default values for properties. These are the settings which are used by default.

The default values may be overridden by selecting the properties menu item on the Workarea Context menu. Setting values at this level sets the properties for all projects and files.

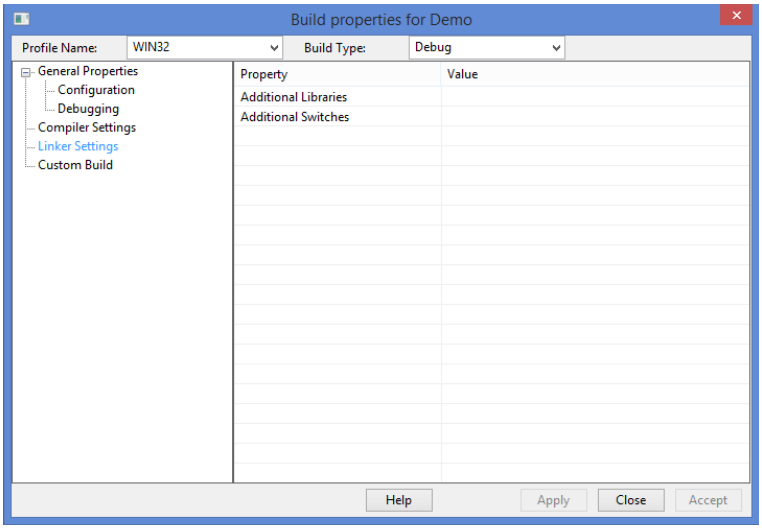
The Workarea properties may be overridden by selecting the properties menu item on the context menu for any project. Setting values at this level sets the properties for the project and all files it contains, but does not affect other projects.

The project properties may be further overridden by selecting the properties menu item on the context menu for any file. Setting values at this level sets the properties for the file but does not affect any other files or projects.

The project properties work internally by utilizing built-in macros, some of which are exposed on the property pages. An example of this is **PROJECTDIR** which expands to the location where the project file is located. The properties system also imports the user's environment variables, so it is possible to define entities outside the IDE and use them as parameters.

At the top of the project properties dialog are selection boxes which allow selection of which Profile the settings apply to. These selection boxes default to the currently selected profile. There are two selection boxes, the 'profile' selection box which names a profile, and the 'build type' selection box which allows a choice between release and debug modes. Note that these selection boxes do not change the currently selected profile, only the profile that new settings will apply to.

On the left hand side of the dialog is a tree view that lists the available pages; on the right hand side properties for a page will be shown. Following is an example of how one of the project properties dialog pages appears.



The Project Properties Dialog has the following panes:

**General Properties**:
General Project Configuration Configuration about the project in general
Debug Project Configuration Configuration about how to debug the project

**Tools Properties**:
Compiler Properties Configuration for the C compiler
Assembler Properties Configuration for the Assembler
Resource Compiler Properties Configuration for the Resource Compiler
Linker Properties Configuration for the Executable File Linker
Librarian Properties Configuration for the Static Library Librarian
Custom Build Properties Configuration for things that happen after the build of a project

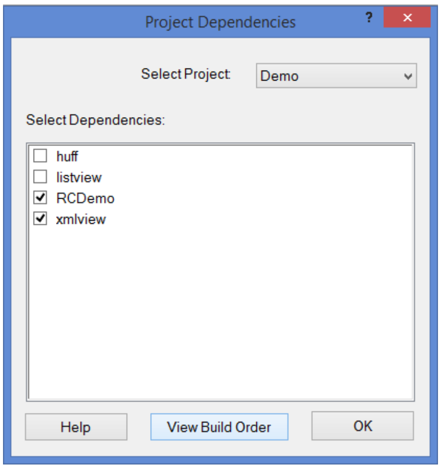
Global setup such as that for the editor is found in the General Properties settings.

## Project Dependencies Dialog

Generally the IDE keeps track of header file dependencies automatically, by parsing source files when something is added or changed and using the embedded #include statements to keep track of the header files the file depends on.

However, when there are multiple projects in the WorkArea some of them may depend on each other. For example, an executable may depend on one more more static libraries.

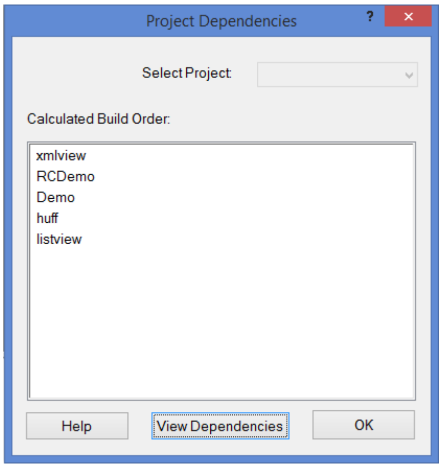
The IDE cannot determine these types of dependencies without help. The Project Dependencies dialog allows you to set these dependencies. It appears as follows:



In this dialog, select the project whose dependencies are to be set in the upper combo box. Then a list of the other projects in the WorkArea will appear and it is possible to select which other projects this project depends on.

Note that sometimes, a project in this list will be grayed out and will not be selectable. This happens because these screens create a dependency tree, and clicking on that item would cause a loop in the tree as that item already depends on the item being worked on. The IDE does not allow this situation to occur.

The dependency tree is eventually flattened. This is to make a list of projects in the order they should be built in order to make sure that dependencies of a project are always built before the project. To view the projects in the order they will be built select 'View Build Order'. The dependencies dialog will then look as follows:



Calculated Build Order:

xmlview
RCDemo
Demo
huff
listview

If a project depends on a static library, the project will be linked against that static library after the library is built.

When done setting dependencies, select 'OK'.

## Profiles

Profiles are a convenient way to allow creation of different groups of settings, and associate them with a profile name.  By selecting a profile, an entire group of settings may be selected, which will then be used in subsequent build operations.  A New Profile may also be added.

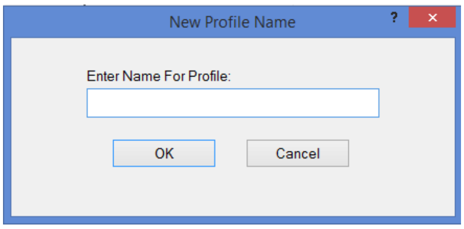
For example, it might be convenient to create a testing profile that selects specific preprocessor definitions to put a project in a test mode.

There are two build types associated with each profile; the release type and the debug type.  The difference is that the release type will use compiler optimizations, and will not have debug information.  The debug type will not use compiler optimizations, but will have debug information suitable for debugging the program.

In the default case the selected profile name and build type are used as part of the directory name which indicates where to put the output files, however, this behavior can be altered through build settings.

## New Profile Dialog

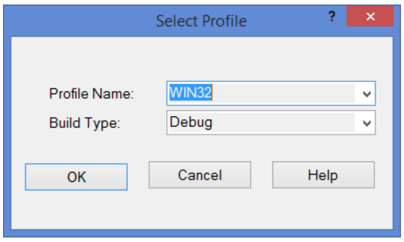
The New Profile dialog is accessible from the Project Properties Dialog.  It allows addition of a new profile name that will allow creating an independent set of settings for a build process.  It appears similar to what follows:



Typing a new name and selecting 'ok' will result in the profile being added to the list of possible profiles.  It will have the default configuration.

## Select Profile Dialog

The Select Profile dialog is accessible by selecting **Select Profile** from the  Project Menu.  It allows selection of which profile will be used for subsequent builds.  It appears similar to what follows:



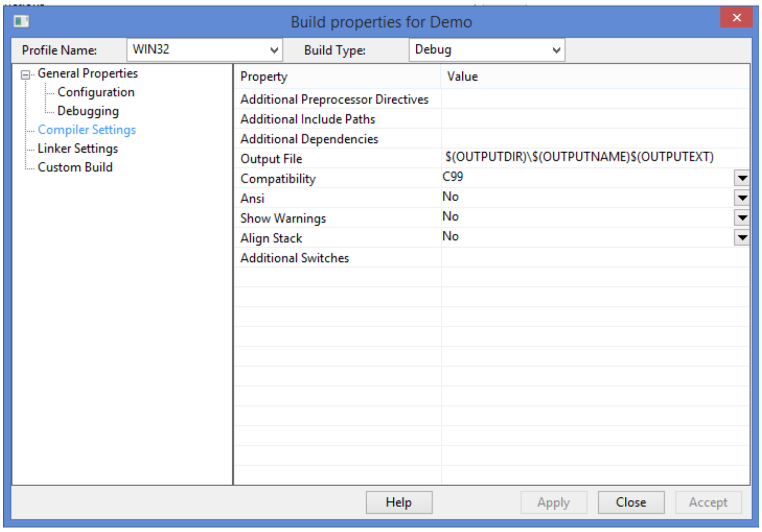
Profile Name: WIN32

Build Type: Debug

The **Profile Name** selection box selects the profile; the **Build Type** selection box selects whether to build in debug or release mode.  Selecting a profile and pressing ok will cause the selected profile to be used in subsequent build operations.

## Compiler Properties

The Compiler Properties page is inside the Project Properties Dialog.  It allows allows modification of project settings related to compiling individual C language source files with the C language compiler.  It appears similar to what follows:



Build properties for Demo

Profile Name: WIN32 Build Type: Debug

| Property | Value |
|---|---|
| Additional Preprocessor Directives | |
| Additional Include Paths | |
| Additional Dependencies | |
| Output File | $(OUTPUTDIR)\$(OUTPUTNAME)$(OUTPUTEXT) |
| Compatibility | C99 |
| Ansi | No |
| Show Warnings | No |
| Align Stack | No |
| Additional Switches | |

The following properties are available on this page:

**Additional Preprocessor Directives** is a space-delimited list of macro declarations. This is a way to duplicate #define behavior from the command line, which allows some definitions to be kept as part of the build process instead of forcing them to reside in header files. The macro definition has the form `macroname=value` or simply `macroname` if it is to be defined without a value. If it is required to put spaces within the value, enclose the entire thing in quotes, for example: "`macroname=my value`".

**Additional Include Paths** is a semi-colon delimited list of paths to search for include files. If relative paths are specified, they will be relative to the directory in which the project file is stored.

**Additional Dependencies** specifies a list of additional dependencies for the file. Normally, the IDE figures out the dependencies and it isn't necessary to use this.

**Output File** is the name of the output file. It is normally set to **$(OUTPUTDIR)\$(OUTPUTNAME)$(OUTPUTEXT).** where **$(OUTPUTDIR)** is derived from the output path, **$(OUTPUTNAME)** is the stem of the source file name, and **$(OUTPUTEXT)** is the literal '.obj'.

**C99** is set to yes if parsing for C99 language constructs should be allowed. Otherwise the compiler will be limited to C89 language constructs.

**Ansi** is set to yes if the compiler should generate an error if one of several commonly used extensions to the ANSI standard is used. For example, in C89 it is common for compilers to allow a comma at the end of a list of enumerations, however, the C89 ansi standard forbids it and turning on ANSI mode will disable the extension.

**Show Warnings** is set to yes if the IDE should show both Warnings and Errors in the Information Window during the build process. Otherwise only Errors will be shown.

**Align Stack** is set to yes if the compiler should generate extra code to keep the processor stack aligned to the nearest 16-byte increment when entering a new function. This option can speed up floating point code that uses 'double' values liberally.

**Additional Switches** selects additional compiler switches which aren't directly supported by the IDE. Any switch that can be specified on the command line may be placed here. For example, putting **+i** here will cause the compiler to generated a preprocessed output file.

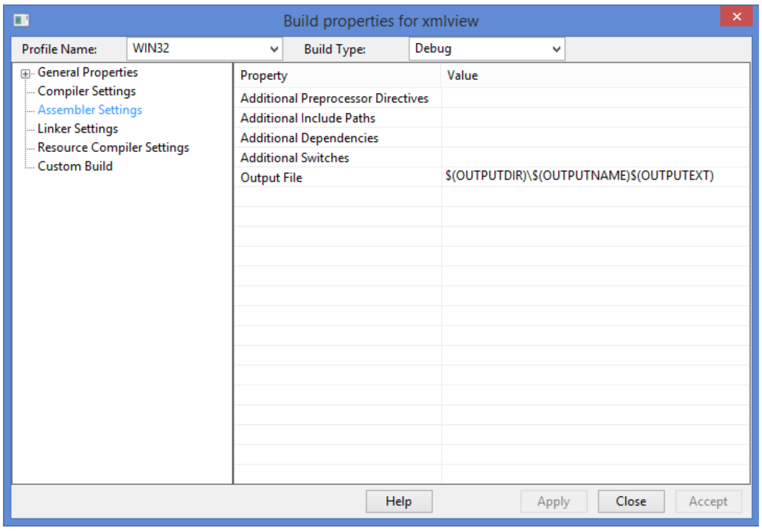
**Apply** saves changes but leaves the dialog open

**Close** closes the dialog without saving changes

**Accept** saves changes then closes the dialog

# Assembler Properties

The Assembler Properties page is inside the Project Properties Dialog. It allows modification of project settings related to compiling individual Assembly language source files with the assembler. It appears similar to what follows:



Build properties for xmlview

Profile Name: WIN32 Build Type: Debug

| Property | Value |
|---|---|
| Additional Preprocessor Directives | |
| Additional Include Paths | |
| Additional Dependencies | |
| Additional Switches | |
| Output File | $(OUTPUTDIR)\$(OUTPUTNAME)$(OUTPUTEXT) |

The following properties are available on this page:

**Additional Preprocessor Directives** is a space-delimited list of macro declarations. This is a way to duplicate %define behavior from the command line, which allows some definitions to be kept as part of the build process instead of forcing them to reside in header files. The macro definition has the form `macroname=value` or simply `macroname` if it is to be defined without a value. If it is required to put spaces within the value, enclose the entire thing in quotes, for example: "`macroname=my value`".

**Additional Include Paths** is a semi-colon delimited list of paths to search for include files. If relative paths are specified, they will be relative to the directory in which the project file is stored.

**Additional Dependencies** specifies a list of additional dependencies for the file. Normally, the IDE figures out the dependencies and it isn't necessary to use this.

**Additional Switches** selects additional compiler switches which aren't directly supported by the IDE. Any switch that can be specified on the command line may be placed here. For example, putting **-UMyMacro** here globally undefines the macro **MyMacro.**

**Output File** is the name of the output file. It is normally set to **$(OUTPUTDIR)\$(OUTPUTNAME)$(OUTPUTEXT).** where **$(OUTPUTDIR)** is derived from the output path, **$(OUTPUTNAME)** is the stem of the source file name, and **$(OUTPUTEXT)** is the literal '.obj'.

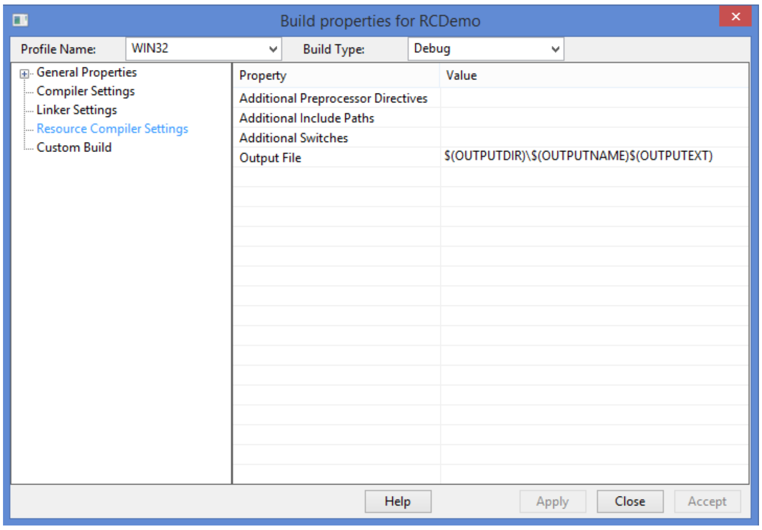
**Apply** saves changes but leaves the dialog open

**Close** closes the dialog without saving changes

**Accept** saves changes then closes the dialog

## Resource Compiler Properties

The Resource Compiler Properties page is inside the Project Properties Dialog. It allows modification of project settings related to compiling individual resource files with the resource compiler. It appears similar to what follows:



Build properties for RCDemo

Profile Name: WIN32 Build Type: Debug

| Property | Value |
|---|---|
| Additional Preprocessor Directives | |
| Additional Include Paths | |
| Additional Switches | |
| Output File | $(OUTPUTDIR)\$(OUTPUTNAME)$(OUTPUTEXT) |

The following properties are available on this page:

**Additional Preprocessor Directives** is a space-delimited list of macro declarations. This is a way to duplicate #define behavior from the command line, which allows some definitions to be kept as part of the build process instead of forcing them to reside in header files. The macro definition has the form `macroname=value` or simply `macroname` if it is to be defined without a value. If it is required to put spaces within the value, enclose the entire thing in quotes, for example: "`macroname=my value`".

**Additional Include Paths** is a semi-colon delimited list of paths to search for include files. If relative paths are specified, they will be relative to the directory in which the project file is stored.

**Additional Dependencies** specifies a list of additional dependencies for the file. Normally, the IDE figures out the dependencies and it isn't necessary to use this.

**Additional Switches** selects additional compiler switches which aren't directly supported by the IDE. Any switch that can be specified on the command line may be placed here.

**Output File** is the name of the output file. It is normally set to **$(OUTPUTDIR)\$(OUTPUTNAME)$(OUTPUTEXT).** where **$(OUTPUTDIR)** is derived from the output path, **$(OUTPUTNAME)** is the stem of the source file name, and **$(OUTPUTEXT)** is the literal '.res.

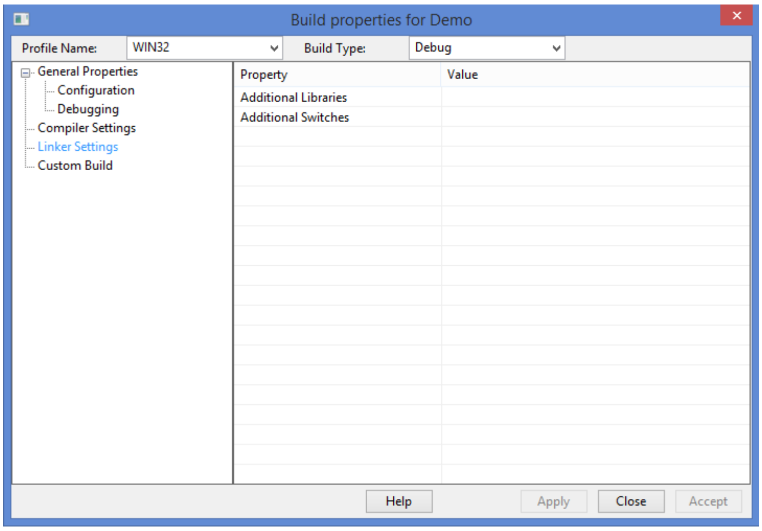
**Apply** saves changes but leaves the dialog open

**Close** closes the dialog without saving changes

**Accept** saves changes then closes the dialog

## Linker Properties

The Linker Properties page is inside the Project Properties Dialog. It allows modification of project settings related to creating executable files. It is only available when the project is some executable type like CONSOLE or GUI. It appears similar to what follows:



Build properties for Demo

Profile Name: WIN32 Build Type: Debug

| Property | Value |
|---|---|
| Additional Libraries | |
| Additional Switches | |

The following properties are available on this page:

**Additional Libraries** is a semi-colon delimited list of libraries to search to resolve code references.

**Additional Switches** selects additional linker switches which aren't directly supported by the IDE.  Any switch that can be specified on the command line may be placed here.  For example **FileAlign:1024** sets the PE file alignment parameter to 1024.

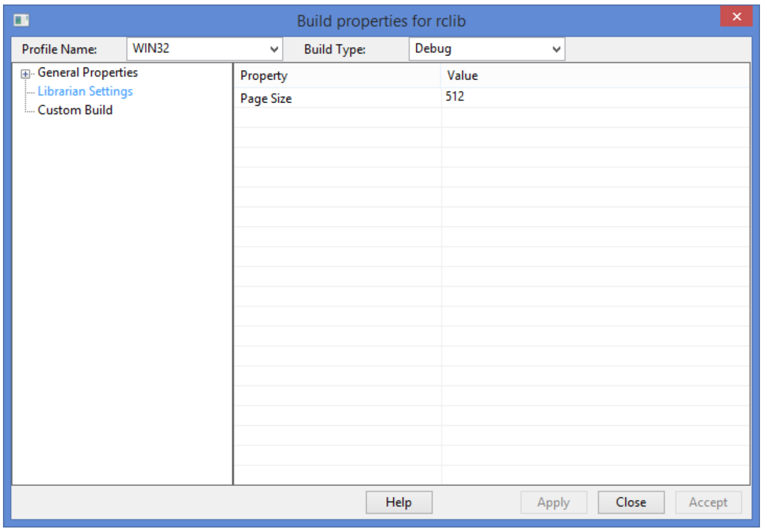
**Apply** saves changes but leaves the dialog open

**Close** closes the dialog without saving changes

**Accept** saves changes then closes the dialog

## Librarian Properties

The Librarian Properies page is inside the Project Properties Dialog.  It allows modification of project settings related to building static libraries.  It is only available when the project type is Static Library.  It appears similar to what follows:



Build properties for rclib

Profile Name: WIN32  Build Type: Debug

| Property | Value |
|---|---|
| Page Size | 512 |

The following properties are available on this page:

**Page Size** gives an internal alignment value.  It is a power of two greater than or equal to 16.  Lower values result in smaller libraries, but the smaller size is also a limitation so sometimes it is desirable to use larger values.  The default is 512.

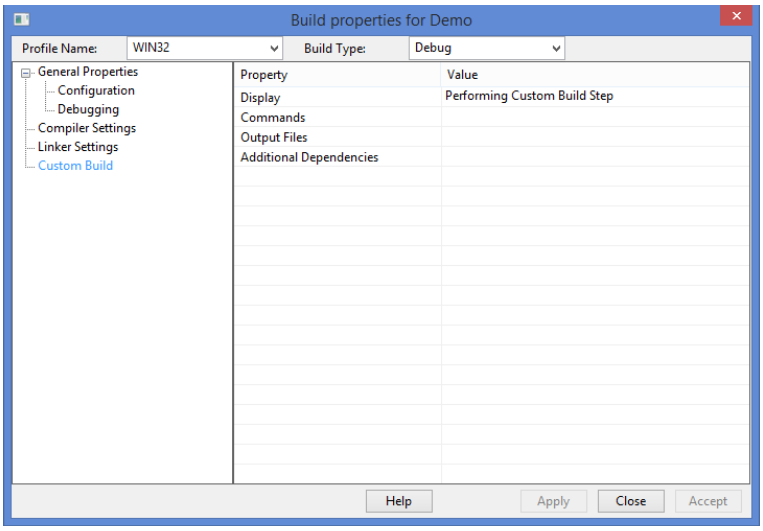
**Apply** saves changes but leaves the dialog open

**Close** closes the dialog without saving changes

**Accept** saves changes then closes the dialog

## Custom Build Properties

The Custom Build properties page are inside the Project Properties Dialog.  It allows defining something that can happen after a project is built.  By default it does nothing.  It appears similar to what follows:



Build properties for Demo

Profile Name: WIN32  Build Type: Debug

| Property | Value |
|---|---|
| Display | Performing Custom Build Step |
| Commands | |
| Output Files | |
| Additional Dependencies | |

The following properties are available on this page:

**Display** is a text string to display in the Information Window when starting this build step.

**Commands** is a list of commands.  Note: this version of the IDE only allows one command to be entered.

**Output Files** is a list of files that will be generated by this build step. Note: this version of the IDE only allows one output file to be entered.

**Additional Dependencies** specifies a list of additional dependencies for this build step. For this type of build step, the IDE cannot figure out any dependencies for the build step, and anything that needs to be pre-built would have to be specified here.

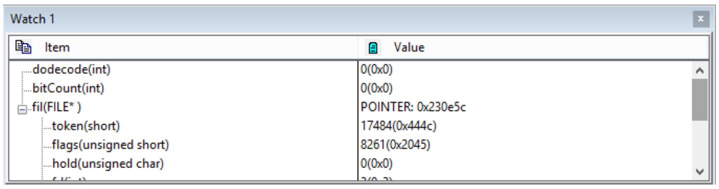
**Apply** saves changes but leaves the dialog open

**Close** closes the dialog without saving changes

**Accept** saves changes then closes the dialog

## Watch Window

The watch window is accessible from the Debug->Windows Menu. It shows the value of variables, and allows them to be modified. It appears similar to what follows:



| Item | Value |
|---|---|
| dodecode(int) | 0(0x0) |
| bitCount(int) | 0(0x0) |
| fil(FILE* ) | POINTER: 0x230e5c |
| token(short) | 17484(0x444c) |
| flags(unsigned short) | 8261(0x2045) |
| hold(unsigned char) | 0(0x0) |

There are four watch windows that may be used independently of each other.

When a value is displayed, clicking on the right-hand side of the value will open a small text window. Typing a new value then pressing ENTER will cause that value to be written to the executable's memory.

Right-Clicking in the watch window opens the Watch Context Menu. This allows adding or removing items to watch.

Right-Clicking in an edit window will display the Editor Context Menu. The **Add To Watch** menu item will display the word under the cursor in the current tab of the window, along with its value.

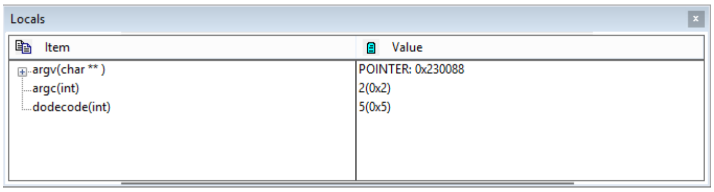
The expression for the watch may be complex, utilizing namespace selectors and structure access operators.

Values that have changed since the last time the program stopped at a breakpoint will be in red; other values will be in black.

The window is updated automatically each time the program stops at a break point.

## Locals Window

The watch window is accessible from the Debug->Windows Menu. It shows the value of variables which are currently in scope, and allows them to be modified. It appears similar to what follows:



| Item | Value |
|---|---|
| argv(char ** ) | POINTER: 0x230088 |
| argc(int) | 2(0x2) |
| dodecode(int) | 5(0x5) |

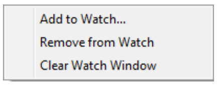
This window is another kind of Watch Window, the difference is that the locals window automatically selects the local (auto) variables which are currently in scope so that they don't have to be individually added to a watch window. In a C++ member function, it will also show the this pointer.

When a value is displayed, clicking on the right-hand side of the value will open a small text window. Typing a new value then pressing ENTER will cause that value to be written to the executable's memory.

The window is updated automatically each time the program stops at a break point.

## Watch Context Menu

The Watch Context Menu is accessed by right-clicking in the Watch Window. It has menu items used for working with the Watch Window. It appears similar to what follows:



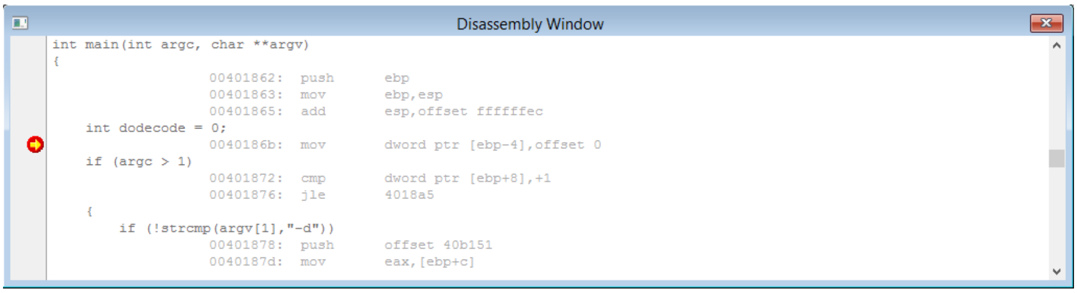
**Add To Watch** opens the Add To Watch Dialog to allow a variable name to be specified. The variable will be displayed along with its value in the current tab of the Watch Window. If the watch window is presently closed it will be opened.

**Remove From Watch** removes the selected item from the Watch Window.

**Clear Watch Window** removes all items from the Watch Window

## Disassembly Window

The Assembly Window is accessed from the Debug->Windows Menu. It shows assembly language instructions for the program. It appears similar to what follows:



Disassembly Window

```
int main(int argc, char **argv)
{
                      00401862:  push      ebp
                      00401863:  mov       ebp,esp
                      00401865:  add       esp,offset ffffffec
    int dodecode = 0;
                      0040186b:  mov       dword ptr [ebp-4],offset 0
    if (argc > 1)
                      00401872:  cmp       dword ptr [ebp+8],+1
                      00401876:  jle       4018a5
    {
        if (!strcmp(argv[1],"-d"))
                      00401878:  push      offset 40b151
                      0040187d:  mov       eax,[ebp+c]
```

Generally the window will start at the position the break point is at.  However if focus is set to the window, then the 'goto' toolbar item (or CTRL-G) will bring up a dialog allowing the address to be changed.  A number can be entered here, or the name of a variable.  It is also possible to type an expression into this box and have it evaluated.

In the window, disassembled instructions will be shown in a light gray and commentary showing the source code will be in a darker gray.

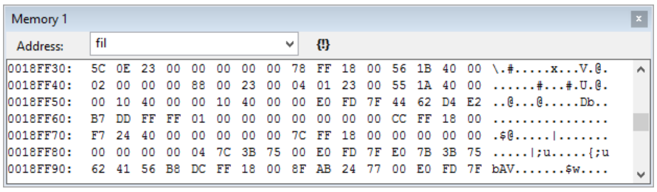
While the window has focus, the up and down arrows and page up and page down keys will allow navigation through the disassembly.  As a convenience, pressing the HOME key will show the executable code address where the program counter is currently stopped.

While the disassembly windows has focus, debugger single-step commands will be for individual lines of disassembled code, instead of for source code file lines.

Clicking on a source code line will bring up an editor window to allow editing the source code.   Clicking on a disassembled instruction will result in a breakpoint being set or cleared.

## Memory Window

The Memory Window is accessed from the Debug->Windows Menu.  It appears similar to what follows:



Memory 1

Address: fil

```
0018FF30:  5C 0E 23 00 00 00 00 00 78 FF 18 00 56 1B 40 00 \.#.....x...V.@.
0018FF40:  02 00 00 00 88 00 23 00 04 01 23 00 55 1A 40 00 ......#...#.U.@.
0018FF50:  00 10 40 00 00 10 40 00 00 E0 FD 7F 44 62 D4 E2 ..@...@.....Db..
0018FF60:  B7 DD FF FF 01 00 00 00 00 00 00 00 CC FF 18 00 ................
0018FF70:  F7 24 40 00 00 00 00 00 7C FF 18 00 00 00 00 00 .$@.....|.......
0018FF80:  00 00 00 00 04 7C 3B 75 00 E0 FD 7F E0 7B 3B 75 .....|;u.....{;u
0018FF90:  62 41 56 B8 DC FF 18 00 8F AB 24 77 00 E0 FD 7F bAV.......$w....
```

There are four memory windows which may be used independently of each other.

The address is shown on the left, followed by a sequence of bytes, followed by a sequence of ASCII values for the byte sequence.  By default 16 bytes will be shown per line, however if the window is resized it will automatically adjust the number of bytes per line to fit.

At the top of the window is a small text box which is used to tell what memory should be viewed.  An address can be entered here, or the name of a variable.  It is also possible to type an expression into this box and have it evaluated.  A toolbar button is used to choose whether the address should remain static or whether it should be re-evaluated every time the program stops at a breakpoint.

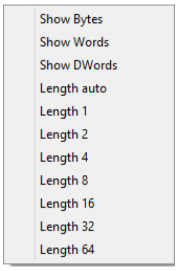
When the memory portion of the window has keyboard focus, a black cursor will appear over one of the values.  The cursor may be moved around to select a value by using the arrow and page keys.  Typing in a Hexadecimal digit will turn the cursor to blue; a second digit can be typed at this time.  While the cursor is blue, the ESCAPE key will void the edit function and return the cursor to black, or the ENTER key will accept the new value and attempt to write it to the program's memory space.  If successful the display will update with a black cursor and show the new value; if the new value cannot be written then a dialog will appear to inform you of the problem.

Values which have changed since stopping at the last breakpoint will be in red; other values will be in black.

The window is automatically updated whenever the program stops at a break point.

## Memory Context Menu

The Memory Context Window is accessed by right-clicking in the Memory Window.  It allows selection of the size of the data displayed and the number of bytes of data to show per line.  It appears similar to what follows:



The following data sizes are allowed:

BYTE - selects a byte per unit (two hexadecimal characters)

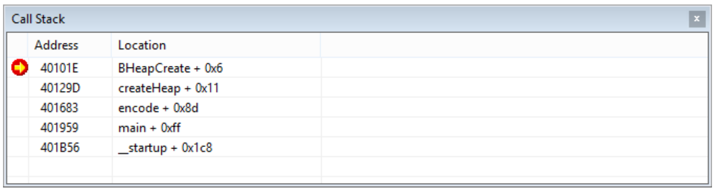
WORD - selects a word per unit (two bytes or four hexadecimal characters)

DWORD - selects a dword per unit (four bytes or eight hexadecimal characters)

The auto selection for bytes per line will adjust the number of bytes shown based on the window width.

## Call Stack Window

The Call Stack Window is accessible from the Debug->Windows Menu.  shows a back trace of all functions that were called on the way to hitting the current break point.  This information is retrieved from a processor location known as the stack.  The window appears similar to what follows:



Call Stack

|  | Address | Location |
|---|---|---|
| ➡ | 40101E | BHeapCreate + 0x6 |
|  | 40129D | createHeap + 0x11 |
|  | 401683 | encode + 0x8d |
|  | 401959 | main + 0xff |
|  | 401B56 | _startup + 0x1c8 |

The window shows the function address and the name when debug information is available.  Sometimes debug information won't be available, such as for code that is stopped in a C language library function, or in the operating system function.

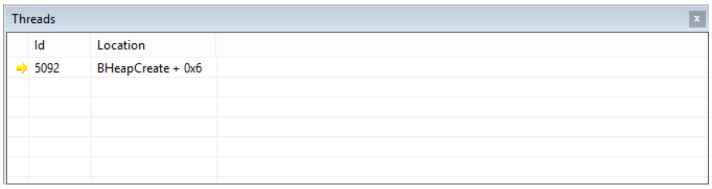
The back trace will be shown for the thread that is selected in the Thread Window.  Changing which function is selected by clicking on it will be reflected in the Thread Toolbar.

Additionally, expression calculations depend on the scope set up by this selection.

The back trace is shown in reverse order, with the most recent function at the top.

## Thread Window

The Thread Window is accessible from the Debug->Windows Menu.  It shows currently active threads.  It appears similar to what follows:



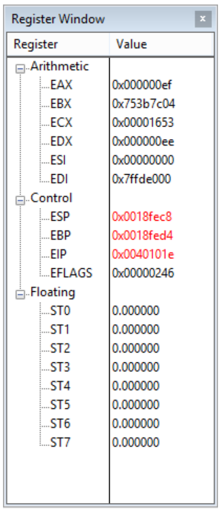
| | Id | Location |
|---|---|---|
| ➪ | 5092 | BHeapCreate + 0x6 |

This window shows the address each thread is at, and the name of the function it is in when available.  The thread that is at a break point is marked with a yellow arrow.

When a new thread is selected by clicking on it, the Thread Toolbar will be updated, and the Call Stack and Register windows will reflect the contents of the selected thread.

## Register Window

The Register Window is accessible from the Debug->Windows Menu.  It shows information about the CPU registers.  It looks similar to what follows:



| Register | Value |
|---|---|
| Arithmetic | |
| EAX | 0x000000ef |
| EBX | 0x753b7c04 |
| ECX | 0x00001653 |
| EDX | 0x000000ee |
| ESI | 0x00000000 |
| EDI | 0x7ffde000 |
| Control | |
| ESP | 0x0018fec8 |
| EBP | 0x0018fed4 |
| EIP | 0x0040101e |
| EFLAGS | 0x00000246 |
| Floating | |
| ST0 | 0.000000 |
| ST1 | 0.000000 |
| ST2 | 0.000000 |
| ST3 | 0.000000 |
| ST4 | 0.000000 |
| ST5 | 0.000000 |
| ST6 | 0.000000 |
| ST7 | 0.000000 |

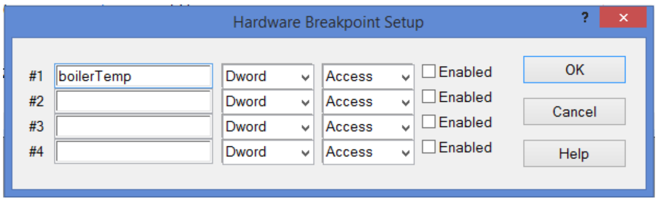
This window has three sections.  The Arithmetic section shows information about the basic register set used for memory and mathematics operations.  The Control section shows information about registers used in CPU control, such as the program counter and stack pointer.  The Floating Point section shows information about the floating point registers.

Register values that changed between the last two breakpoints are shown in red; values that did not change are shown in black.  Clicking on a register value allows it to be edited.

The register values shown will be for the thread selected in the Thread Window.

## Hardware Breakpoints Dialog

The Hardware Breakpoints Dialog is accessible by selecting **Hardware Breakpoints** from the Debug Menu.  It is used to set special microprocessor breakpoints that can be triggered on data access.  It appears similar to what follows:



| | | | | |
|---|---|---|---|---|
| #1 | boilerTemp | Dword | Access | □ Enabled |
| #2 | | Dword | Access | □ Enabled |
| #3 | | Dword | Access | □ Enabled |
| #4 | | Dword | Access | □ Enabled |

Hardware breakpoints are somewhat limited.  There are only four of them, and they can only be set on values that are integers or smaller.  Data Breakpoints can often be a better approach, however, hardware breakpoints do not slow down the program like data breakpoints do.

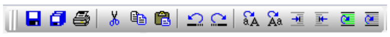
Each Hardware Breakpoint has an associated name box, which holds an Expression that determines the address the break point starts at.

The breakpoints can be configured for an access **Size** in 1, 2, or 4 byte increments and set to an **Access Type** of either Write or Access.   The Write mode causes the program to stop of data is written to the specified address.   The Access mode causes the program to stop if the data is either read or written.

Additionally, the break point may be disabled to prevent it from triggering.

## Edit Toolbar

The Edit toolbar has controls which are commonly used when working with source files.  It may be customized using the Toolbar Customization Dialog.  In the undocked state, and when all buttons are in their normal position, it appears as follows:



The Edit toolbar has the following options:

**Save** Save the current file to disk
**Save All** Save all open files to disk
**Print** Open the Printer Dialog in preparation for printing the file.

**Cut** Cut the selection and put it on the clipboard
**Copy** Copy the selection to the clipboard
**Paste** Paste the clipboard at the current cursor position
**Undo** Undo the last thing done
**Redo** Redo the last thing undone

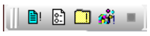
**To Upper Case** the selected text in the edit window is converted to upper case
**To Lower Case** the selected text in the edit window is converted to lower case

**Indent** Move the selected text to the right by one tab space
This may uses either spaces or tabs depending on the Editor Formatting page.
**Unindent** Move the selected text to the left by one tab space. Once text hits the left margin it does not move backward any further.

**Comment** Add comment markers to the beginning of each line in a block of selected text.
**Uncomment** Remove comment markers which are at the beginning of lines in a block of selected text.

## Build Toolbar

The Build toolbar has controls for things that are commonly done while compiling and linking programs. It may be customized using the Toolbar Customization Dialog. In the undocked state, and when all buttons are in their normal position, it appears as follows:



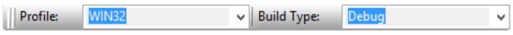
The Build toolbar has the following options:

**Compile File** Compile the current edit window
**Build All** Build out-of-date portions of all projects
**Build Active** Build out-of-date portions of the active project
**Rebuild All** Rebuild every part of every project

**Stop Build** Stop the entire build

## Build Type Toolbar

The Build Type toolbar has two windows that allows choice of the profile and the build type. It may be customized using the Toolbar Customization Dialog. In the undocked state, and when all buttons are in their normal position, it appears as follows:

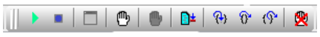


Profile: WIN32 Build Type: Debug

The Thread toolbar has the following options:
**Profiles** a combo-box which has a list of existing profiles. Selecting one sets the build profile.
**Build Type** a combo-box which allows the choice of building in either debug or release mode.

## Debug Toolbar

The Debug toolbar has controls which are useful during debugging of a program. It may be customized using the Toolbar Customization Dialog. In the undocked state, and when all buttons are in their normal position, it appears as follows:



The Debug toolbar has the following options:

**Start/Continue debugging** Start debugging, or run the program starting at the current breakpoint if it is already started.

If the debugger is being started, Debugging Windows are opened. The program will stop at a break point, or at the *main()* or *WinMain()* function if no break points are set.

**Stop Debugging** Stop debugging and close the Debugging Windows.

**Run Without Debugger** Run the program without debugging it.

**Toggle Breakpoint** Toggle Break Point at the current line in the currently selected edit window.

**Stop Program** If the program is running, interrupt it and make it stop as if it hit a breakpoint.

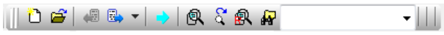
**Run** Run the program. It will stop at the next break point encountered, or when the program exits.
**Run To Cursor** Run the program. It will stop at the next break point encountered, when the program exits, or when the program flow reaches the cursor position in the current active edit window.

**Step In** Step into a subroutine and stop at the beginning.
**Step Over** Step over a subroutine and break after the subroutine is done.
**Step Out** Step out of the current function to the next position in the next enclosing function.

**Remove All Breakpoints** Delete all breakpoints to let the program run unobstructed.

## Navigation Toolbar

The Navigation toolbar has controls which are commonly used to create windows with source files, or navigate through existing windows and/or source files. It may be customized using the Toolbar Customization Dialog. In the undocked state, and when all buttons are in their normal position, it appears as follows:



The Navigation toolbar has the following options

**New** Create a new text file
**Open** Prompt for a file name to open, then open it in the edit window

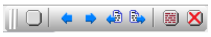
**Back** Browse backwards through recently accessed files
**Forward** Browse forwards through recently accessed files
**Goto** Open the Goto Line Dialog to allow positioning to a specific line within the selected file

**Find** Open the Find/Replace Dialog to locate text in windows
**Find Next** Locate the next occurrence of text
**Replace** Open the Find/Replace Dialog to locate text in files or windows and replace it
**Find In Files** Open the Find/Replace Dialog to locate text within a group of files

Additionally the Navigation Toolbar has a dropdown combo box that can be used to search for text.  This form of search will search for simple text, in the current window.

# Bookmark Toolbar

The Bookmark toolbar has controls which are used to bookmark text in windows and navigate between bookmarks.  It may be customized using the Toolbar Customization Dialog.  In the undocked state, and when all buttons are in their normal position, it appears as follows:

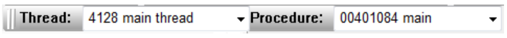


The Bookmark toolbar has the following options:

**Toggle Bookmark** enable a bookmark at the current cursor position, or disable it if it is enabled.
  a bookmark indicater will show up to the left of the line if it is enabled.
**Previous Bookmark** navigate to the previous bookmark in the list
**Next Bookmark** navigate to the next bookmark in the list
**Previous File With Bookmark** navigate to the first previous bookmark in the first previous file in the list
**Next File With Bookmark** navigate to the first next bookmark in the first next file in the list
**Show Bookmarks** show the Bookmarks Dialog to allow selecting a bookmark
**Remove All Bookmarks** remove all bookmarks from all files

# Thread Toolbar

The Thread toolbar has two windows that allow navigation between active threads while debugging.  It may be customized using the Toolbar Customization Dialog  In the undocked state, and when all buttons are in their normal position, it appears as follows:



Thread: 4128 main thread
Procedure: 00401084 main

The Thread toolbar has the following options:

**Threads** a combo-box which has a list of all running threads.  Selecting one sets the active thread to it and rese breakpoints window with the call stack of that thread.
**Breakpoints** a combo-box which has a list of the call stack of the selected thread.  Selecting one opens a window with a file if it is not already open, and set the cursor to the breakpoint.

The Thread and Call Stack windows will be updated with the positions selected in these combo boxes.  The contents of the Call Stack and Register windows will be updated if selections are made on this toolbar.

Additionally, expression calculations depend on the scope set up by this selection.

# Expressions

When adding a variable to the watch window with the watch window dialog, or specifying an address in the memory window edit control, it is possible to type expressions into the dialog and have them evaluated.

The expression evaluator knows most of the operators of the C language.  It is also possible to type-cast expressions, to tell the watch window how to display the results of expressions.

In general, when a variable name is typed into the dialog, that is taken to mean the address of the variable.  However when variables are used in expressions, what happens depends on the type of the variable.  If it is a basic type such as *int* or *short,* the value is fetched and used in the expression, and the result becomes a constant.  However if it is a structured type such as *struct* or *union*, or an array value, the address is still used in the expression.  Normal C language rules apply for such things; for adding constants to structure addresses results in moving further along in an array of the structures, instead of simply offsetting the base address of the structure.

The expression evaluator will handle typecasting of a variable name or address.

Sometimes an expression requires a scope, for example a variable name may appear in several different functions, each of which has called another on the way to reaching the current stop to execution flow.  In general the IDE will use the scope specified by the Thread and Call Stack windows, or from the current cursor position in the case of debugger hints.   It will select from a list of global variables if it cannot locate sometihng in the local scope.

Each expression will be completely re-evaluated each time the program stops at a break point.

# Regular Expressions

Regular expressions may be used in the query boxes of the Find/Replace dialog.  The regular expression operators are a subset of those used by the EMACS editor.  In general characters have their normal meanings, but there are a few special characters that can be used.

Regular expressions work by taking a list of operators with generic meanings and trying to match parts of the source text (the edit file).  If all the operators taken together in sequence match some string in the source text, the search operation is successful.

. matches any single character
+ matches one or more occurrences of the preceding character
* matches zero or more occurrences of the preceding character
? matches exactly zero or one occurrences of the preceding character
^ matches beginning of line
$ matches end of line
[ and ] define a list of characters which can match present character
  between the brackets can be placed either a list of letters, any of which matches the current character, or the name of a character class which is matched against the current character.  The format for a character class is:

[:classname:]

for example [[:alpha:]] matches any letter.  The class names follow the macros found in ctype.h:  the following class names are valid:

alpha matches a letter
upper matches an upper case letter
lower matches a lower case letter
digit matches a digit
alnum matches a letter or digit
xdigit matches a hexadecimal digit
space matches any white space character, e.g. spaces,
  newlines, tabs, form feeds
print matches any printable character
punct matches punctuation characters
graph matches non-space printable characters
cntrl matches control characters
blank matches space and tab

| alternator, an 'or' operator

\ escape character.  When it precedes a special character, it quotes that character.  When used with other characters it can cause special matching functions to occur

The escape sequences are as follows:

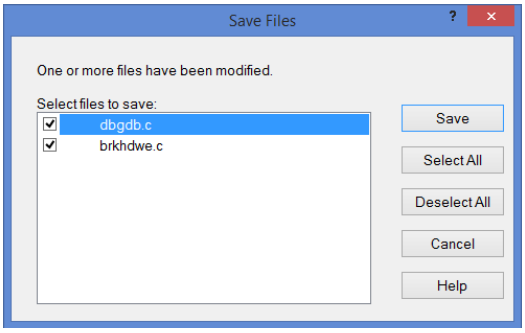
\b matches a word boundary
\B matches inside a word
\w matches word constituents
\W matches non-word constituents
\< matches beginning of word
\> matches end of word
\` matches beginning of buffer
\' matches end of buffer
\( and \) define a group to be indexed in back-reference and replace operations
\{ and \} interval operators
  for example: "a{2,4} matches from 2 to 4 'a' characters.  less than two or greater than four characters don't match
\digit back-reference operator... references a parenthesized group

In the replace field most of the above sequences have no effect.  The following sequences have special meaning:

\digit reference a parenthesized group specified in the find window
\ quote the next character

## Save File Dialog

When exiting the IDE, the IDE will check if any files in the edit windows have been modified but not saved.  If so a dialog similar to the following is displayed to the following to give the option to choose files to save:



One or more files have been modified.

Select files to save:

☑ dbgdb.c
☑ brkhdwe.c

Clicking on the checkbox next to a file will select or deselect it.  If a save operation is performed, only selected files will be saved.  By default all files that have changed are marked as selected, and files that have not changed are not shown in this dialog box.

**Save** saves the contents of the selected files to disk.  If no files are selected, save does nothing and the dialog closes.

**Select All** marks all files as selected.

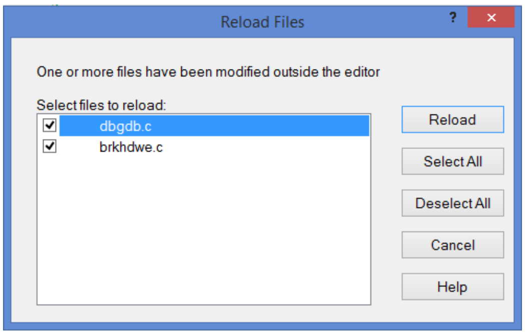
**Deselect** all marks all files as unselected.

**Cancel** closes the dialog and aborts the exit.

**Help** displays this text.

## Reload File Dialog

When the IDE is brought to the foreground, it will check if the contents of any of the files in the edit windows have been changed outside the source editor, for example in another editor or as a result of running a program that writes to the file.  If any files have changed and potentially need to be reloaded, a dialog similar to the following is displayed:



One or more files have been modified outside the editor

Select files to reload:

☑ dbgdb.c
☑ brkhdwe.c

Clicking on the checkbox next to a file will select or deselect it.  If a reload operation is performed, only selected files will be reloaded.  By default all files that have changed are marked as selected, and files that have not changed are not shown in this dialog box.

**Reload** reloads the contents of the selected files from disk.  If no files are selected, reload does nothing and the dialog closes.

**Select All** marks all files as selected.

**Deselect** all marks all files as unselected.

**Cancel** closes the dialog without reloading any files

**Help** displays this text.

## Hot Keys

Many of the IDE functions may be performed with alternate key sequences.  This page gives a list of the key sequences and what they do.

In general, the following key sequences are always in effect:

**CTL-F** Open the Find/Replace Dialog to search for text
**CTL-G** Open the Goto Line Dialog to position text in the current edit window to a specific line
**CTL-H** Open the Find/Replace Dialog to search for and replace text
**F1** The index page of this help file
**F2** Position to the next bookmark

**SHIFT-F2** Position to the previous bookmark
**CTL-F2** Open the Bookmark Dialog to select a bookmark to position to
**ALT-F2** Toggles bookmark status for the current line
**F3** Search for text again (find next)
**SHIFT-F4** Tiles the windows horizontally
**CTL-F4** Tiles the windows vertically
**F5** Start the debugger and/or run the program
**SHIFT-F5** Stop the debugger
**CTL-F5** Run the program until control flow reaches the selected line in the current editor window
**ALT-F5** Stop the program
**F7** Build the parts of all projects which are not up to date
**SHIFT-F7** Build the parts of the active project which are not up to date
**CTL-F7** Compile file in current window
**F9** Toggle break point at the selected line in the current editor window
**F10** Step over the current line
**SHIFT-F10** Run to cursor
**F11** Step into the current line
**ALT-F11** Step out of the current function
**F12** Browse to the definition of the selected word
**SHIFT-F12** Browse back to the previous text position

In addition the following sequences have special meaning in the Editor Window:

**ESC** Close the find window or a code completion window
**SHIFT-CTL-#** # is a digit from 1 - 9.   Mark a position in the text
**CTL-#** # is a digit from 1 - 9.   Goto a previously marked position in the text
**CTL-C** Copy selection to clipboard
**CTL-L** Page Break
**CTL-R** Redo the last undone operation
**CTL-S** Save the file in the current edit window
**CTL-T** Delete word to right of cursor
**CTL-V** Paste clipboard to cursor position.  This overwrites the selection if there is one
**CTL-X** Copy the selection to the clipboard, and cut it from the text
**CTL-Y** Delete line cursor is on
**CTL-Z** Undo the last operation
**ALT-{** Navigate to beginning of block when cursor is on a '}'
**ALT-}** Navigate to end of block when cursor is on a '{'
**CTL-[**                                   Open full-size edit window
**CTL-]**                                   Close full-size edit window
**CTL-|** Place the line the cursor is at in the center of the window (vertical center)
**SHIFT-F1** Show run time library help for the word under the cursor.
**CTL-F1** Show WIN32 help for the word under the cursor
**CTL-F7** Compile the file in the source window
**BACKSPACE** Delete the character preceding the cursor
**DELETE** Delete the character at the cursor
**END** Move the cursor to the end of the line
**CTL-END** Move the cursor to the end of the file
**HOME** Move the cursor to the beginning of the line
**CTL-HOME** Move the cursor to the beginning of the file
**INSERT** toggles the insert mode between INSERT and OVERWRITE
**PAGE DOWN** Move the cursor down one page
**PAGE UP** Move the cursor up one page
**TAB** Insert a tab character, or indent text if it is selected.  Depending on the Editor Formatting Configuration, the tab may be replaced by enough spaces to reach the next tab position.
**SHIFT-TAB** Delete a tab or spaces to the left of the cursor, or unindent if text is selected
**DOWN ARROW** Navigate downward one line
**UP ARROW** Navigate upward one line
**LEFT ARROW** Navigate left one character
**RIGHT ARROW** Navigate right one character
**CTL-RIGHT ARROW**  Navigate right one word
**CTL-LEFT ARROW**  Navigate left one word

The following keys have special meaning in the Workarea Window:

**DOWN ARROW** Navigate downward one line
**UP ARROW** Navigate upward one line
**LEFT ARROW** Navigate left one character
**RIGHT ARROW** Navigate right one character
**INSERT** If the selection is on a project, open the Open File Dialog and prompt for a new project to insert prior to this one.  Otherwise open the Open File Dialog and prompt for new files to insert in the selected project.
**CTL-INSERT**   Expand a node
**DELETE** If the selection is on a project, a prompt will appear asking whether to remove it from the workarea.  Accepting it will remove the project.  If the selection is on a file, the file will be removed from the workarea.
**CTL-DELETE**  Collapse a node
**ENTER**  open the file corresponding to the node

The following keys have special meaning in the Assembly Window:

**CTL-G** Prompt for an address to position to
**HOME** Navigate to the position the program is currently stopped at
**DOWN ARROW** Navigate downward one line
**UP ARROW** Navigate upward one line
**LEFT ARROW** Navigate left one character
**RIGHT ARROW** Navigate right one character
**PAGE-UP** Navigate upward one page
**PAGE-DOWN** Navigate downward one page

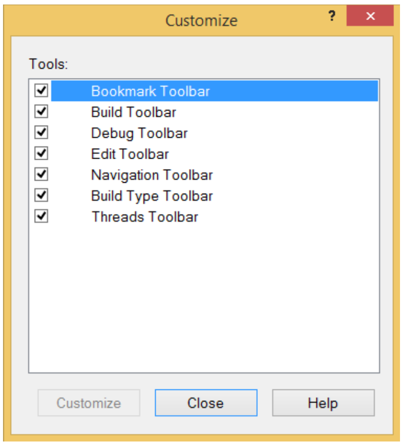
The following keys have special meaning in the Memory Window:

**DOWN ARROW** Navigate downward one line
**UP ARROW** Navigate upward one line
**LEFT ARROW** Navigate left one character
**RIGHT ARROW** Navigate right one character
**PAGE-UP** Navigate upward one page
**PAGE-DOWN** Navigate downward one page
These keys allow navigation through the memory
**0-9, A-F** These keys let you enter a new value.  The characters will turn blue until ENTER is pressed to accept the change
**ENTER** Accept a new value and program it into memory
**ESCAPE** Restore the original value for a change that has not been accepted

Most of the debug windows allow use of **CTL-C** to copy text from the window onto the clipboard.

## Customize Tools

The Customize Tools Dialog is accessible from the Tools Menu.  It shows a list of toolbars.  The dialog appears as follows:



Tools:

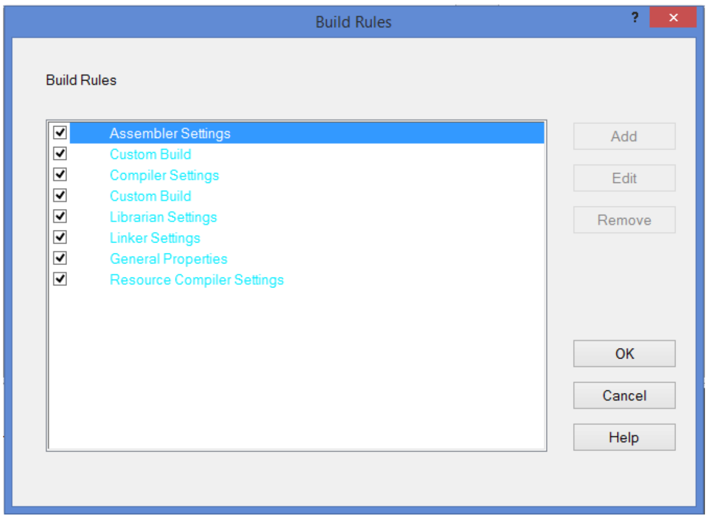
- ☑ Bookmark Toolbar
- ☑ Build Toolbar
- ☑ Debug Toolbar
- ☑ Edit Toolbar
- ☑ Navigation Toolbar
- ☑ Build Type Toolbar
- ☑ Threads Toolbar

Here each of the six toolbars is listed, with a checkbox to indicate whether the toolbar is selected or deselected. Selected toolbars appear on the display; deselected toolbars do not.

Pressing the Customize button will open the Toolbar Customization dialog, which allows rearranging the buttons on the toolbar.

## Build Rules

The Build Rules dialog allows disabling internal build rules. In a future version of the IDE it will also allow creating and editing new build rules. It appears as follows:

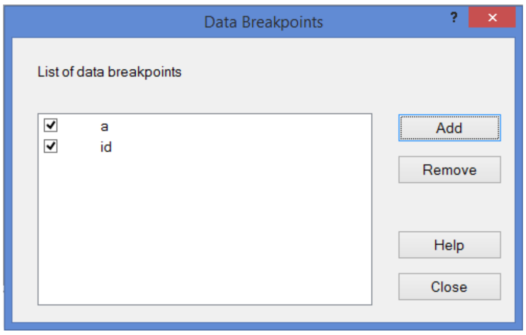


Build Rules

- ☑ Assembler Settings
- ☑ Custom Build
- ☑ Compiler Settings
- ☑ Custom Build
- ☑ Librarian Settings
- ☑ Linker Settings
- ☑ General Properties
- ☑ Resource Compiler Settings

## Data Breakpoint

The Data Breakpoints Dialog is used to set special microprocessor breakpoints that can be triggered on data access. It appears as follows:



List of data breakpoints

- ☑ a
- ☑ id

Data Breakpoints are general purpose breakpoints. They can be set to trigger when any variable is accessed, including a complete structure or an array. However, a data breakpoint slows down the program by a small amount. In some cases where the limits of Hardware Breakpoints can be tolerated, they are a better choice.

Each Data Breakpoint is simply an Expression that determines the address the break point starts at.

Data Breakpoints can be added and removed through the dialog. Additionally, a checkbox to the left of the breakpoint allows the possibility of disabling breakpoint while keeping it in the list.

## Jump List

The Jump List is a pair of combo boxes that appear just below the source file tabs, and above the edit windows which hold source files. It allows selection of a type of global variable in one combo box; when the type of the variable is selected the other combo box will be populated with variables of that type. Selecting a variable from the second combo box causes the IDE to position the cursor to that function.

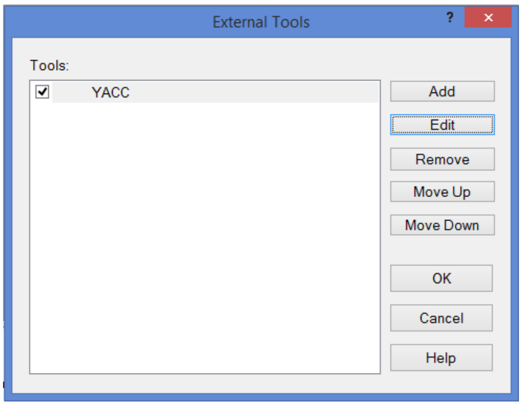
The possible types are:

**Global Functions**
**Global Variables**
**Global Types**
**Preprocessor Macros**

Note that the Jump List depends on the program having previously been compiled with Browse Information enabled on the Hints and Code Completion properties page.

## Custom Tools

The Custom Tools dialog is accessible from the Tools Menu. It allows configuration of programs that can be run directly from the IDE. It appears similar to what follows:



Tools:

☑ YACC

This dialog shows a list of custom tools that have previously been defined. Each tool has a checkmark, which indicates whether the tool should show up in the Tools -> External Tools menu. When a tool appears in the Tools menu, clicking on it will execute the program associated with the tool

The items will show up in the Tools menu in the same order they are displayed in this dialog, so there **Move Up** and **Move Down** options that move the selected item around.

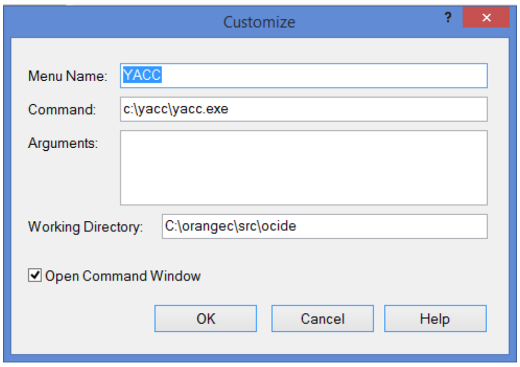
**Add** creates a new item and opens the Custom Tools Editor Dialog to set it up.

**Edit** opens the Custom Tools Editor dialog to edit the selected item.

**Remove** removes the selected item from the list of tools.

## Custom Tools Editor

The Custom Tools Editor is accessible from the Tools Menu. It allows modifying the name of the executable a Custom Tool menu item will run, as well as parameters. It appears similar to what follows:



Menu Name: YACC

Command: c:\yacc\yacc.exe

Arguments:

Working Directory: C:\orangec\src\ocide

☑ Open Command Window

When a custom tool is selected from the Tools Menu, the IDE will use the information specified in this dialog to run the tool.

Several text boxes appear on the dialog:

**Menu Name** name of the item as it will appear on the menu<br>
**Command** name of the program to run. The PATH environment variable will be searched for the program.<br>
**Arguments** any command-line arguments to be provided when the program runs.<br>
**Working Directory** the directory the program will see if it calls a function like `getpwd` or `GetCurrentDirectory`

Additionally, the **Open Command** Window checkbox indicates whether the IDE should open an operating system command prompt to run the program in.

## Resource Editor

OCIDE has a built-in resource editor. This editor is capable of editing the following types of resources:

- Accelerators
- Bitmaps
- Cursors
- Dialogs and Controls
- Icons
- Menus
- RCData
- String Tables
- Version Data

Additionally, you can add Font and Message Table resources although you cannot edit them.

See here for general remarks that may be useful in diagnosing problems loading resources.

A main Resource Window shows a list of all available resources and allows easy addition and removal of resources. Clicking on a resource will bring up the associated resource editor for that resource.

The resource editor interacts with a resource (.RC) file, where it stores the data in ASCII format. It is capable of loading and saving most resource files, although there are a few limits. For an existing resouce file, it will read in most types of resources even if the resource editor does not have native support for editing them.

## Remarks about Resources

The way resource files work, it is possible to reference a resource either by name, or by number. For example for the resource statement:

ID_BMP bitmap "somefile.bmp"

It is not immediately clear whether you would use a statement such as

HBITMAP myBmp = LoadBitmap(hDC, "ID_BMP");

or a statement such as:

HBITMAP myBmp = LoadBitmap(hDC, MAKEINTRESOURCE(ID_BMP));

to actually load the bitmap into the program. One of these is going to work, and the other isn't.

The rule is that if ID_BMP has not been defined anywhere, the resource compiler will compile a 'named' reference to the resource. For example the first example would be the way to load it. However, if it has been defined somewhere:

#define ID_BMP 100
ID_BMP bitmap "somefile.bmp"

Then the resource compiler will compile a numeric reference to the resource, and one would have to use the second way of loading it.

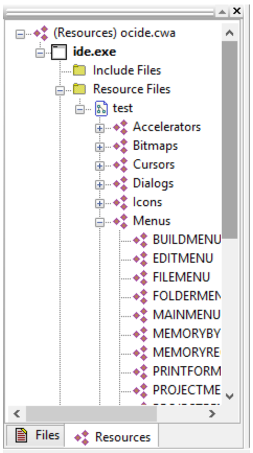
Some projects have defined the resource name, and some have not. So either method may be necessary for loading existing resources depending on the situation. However, this IDE always adds a #define for resources, so the second method using MAKEINTRESOURCE is the preferred method for loading new resources. This entire behavior is well-defined in windows and will be compatible across development environments, except in the case where a development environment adds a #define after the fact without indicating it.

This IDE will load and save resources identifiers in either format; if there is no #define one won't be added. It is only when creating a new resource with the resource editor that one has to be aware to use the numeric reference via MAKEINTRESOURCE instead of the named reference. But this behavior does make it possible to use the named version of resources. Simply delete the associated #define after creating the resource, then reload the resource file.

# Resource Window

The resource window appears in the same view as the Workarea. A tab at the bottom selects it, or selecting an .RC file will automatically select it. Before the resource window will show resources, a resource file must be added to the project. This can be done either by adding a new file with the New File Dialog, and selecting its type to be .rc, or by bringing an existing resource file into the IDE with the Open File Dialog. Once the file is in the project, double clicking on it will load it into the IDE and its list of resources will automatically appear in the resource window.

The resource window appears as follows:



In this case there is only one resource, a dialog with the resource identifieir ID_TEST. The IDE will also create a file RESOURCE.H, which will have resource and control identifiers used by the program. It can be #included into program modules to allow access to the resources.

At this point, right-clicking on the resource file name, which is called RCDemo.rc in this example, will bring up the New Resouce Dialog. This allows selection of a new type of resource to add. When the resource is added it will show up in the tree. If there were no other resources of that type previously, OCIDE will insert a header name, for example 'Dialogs' or 'Accelerators'.

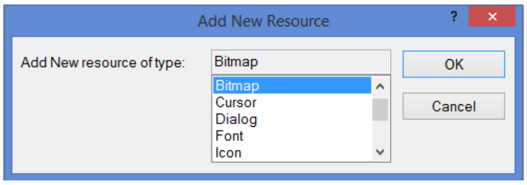
A new resource will be given a default resource identifier, the resource identifier being the handle the C program uses to access the resource. The resource identifier may be renamed either by double-clicking on it in the tree or by going to the property window and editing the resource identifier.

Right-clicking on one of the headers will allow creation of new resources of the type specified by the header. Right clicking on a resource will allow removing the resource.

Resource identifiers in OCIDE are always given a numeric value when created, however, OCIDE does allow you to use string versions of the resource identifiers simply be removing the associated #define from the RESOURCE.H file.

# Add New Resource Dialog

The add new resource dialog is accessible by selecting New Resource from the Resource Menu, or by clicking on an item in the Resource Window. It allows selection of the type of new resource to add. It appears similar to what follows:



It allows adding one of the following types of resources:

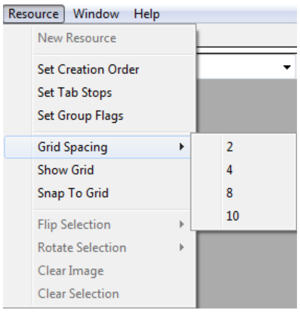
- Accelerators
- Bitmaps
- Cursors
- Dialogs and Controls
- Fonts
- Icons
- Menus
- Message Tables
- RCData

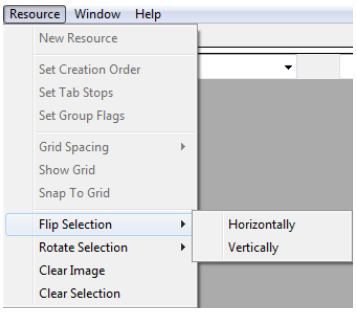
- String Tables
- Version Data

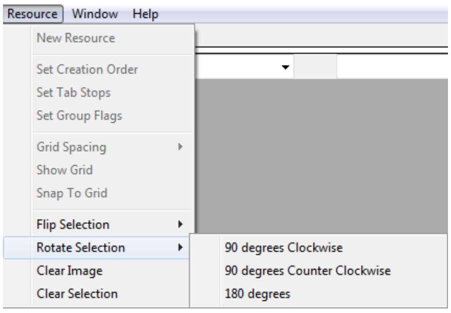
Attempting to add a Bitmap, Cursor, Font, Icon, or Message Table will bring up the Open File Dialog to ask for a file name.  In the case of images, if the file does not exist you will be asked if you want to create it.

# Resource Menu

The resource menu looks as follows:







The first image shows the resource menu as it appears when a dialog edit window is selected.  The following options are available:

**Set Creation Order** the dialog editor will go into the 'Set Creation Order' mode.  In this mode, selecting the controls one by one will set the order the controls will be created.  This affects the order for tabbing through the windows, and it will affect visibility if one window overlaps another.

**Set Tab Stops** the dialog editor will go into the 'Set Tab Stops' mode.  In this mode, clicking on each control will set it's 'Tab Stop' setting.  The 'Tab Stop' setting indicates whether hitting tab on the preceding control will cause this control to get focus.  Each control will be outlined in red if it has no tab stop setting, or blue if it does.

**Set Group Flags** the dialog editor will go into the 'Set Group Flags' mode.  In this mode, clicking on each control will set it's 'Group Flag' setting.  The 'Group Flag' setting indicates whether this control will be in the same group as adjacent controls.  The group flag setting is typically used for things like a set of radio buttons, each of which excludes the others when pressed.  Each control will be outlined in red if it has no group flag setting, or blue if it does.

**Grid Spacing** this sets the density of the grid, when it is shown

**Show Grid** show a grid to help align controls

**Snap To Grid** when a control is created or moved, make sure its upper right corner aligns with the grid.

In the second and third images the menu appears as it would when an image edit window for a bitmap, cursor, or icon is selected.  The following basic options are available:

**Flip Selection**
**Horizontally** flips the selection horizontally
**Vertically** flips the selection vertically

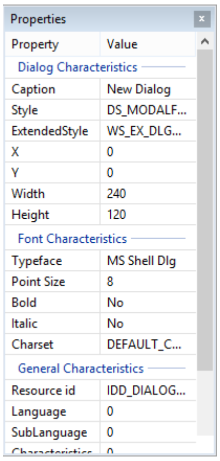
**Rotate Selection**
**90 degrees clockwise** rotates the selection 90 degrees clockwise
**90 degress counterclockwise** rotates the selection 90 degrees counterclockwise
**180 degrees** rotates the selection 180 degrees

**Clear Image** set the image to the background color for bitmaps, or to see through for icons and cursors

**Clear Selection** set the selection area to the background color for bitmaps, or to the see through color for icons and cursors

# Resource Properties

The resource Properties window is in the same view as the Dialog Control Toolbox.  It is accessible by selecting **Property Window** from the View Menu.
Each type of resource has a properties window, which allows setting various attributes of the resource.  This can include things such as a title for dialogs, fonts to use, and various other attributes of resources or controls.  While it is beyond the scope of this documentation to talk about every property that may be edited in depth, an example for a dialog resource is shown below.



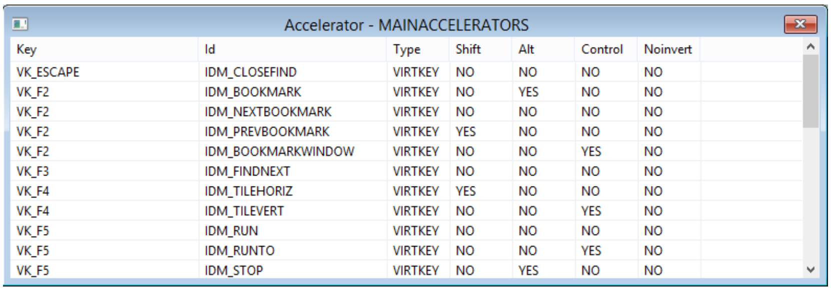
Properties

| Property | Value |
|---|---|
| Dialog Characteristics | |
| Caption | New Dialog |
| Style | DS_MODALF... |
| ExtendedStyle | WS_EX_DLG... |
| X | 0 |
| Y | 0 |
| Width | 240 |
| Height | 120 |
| Font Characteristics | |
| Typeface | MS Shell Dlg |
| Point Size | 8 |
| Bold | No |
| Italic | No |
| Charset | DEFAULT_C... |
| General Characteristics | |
| Resource id | IDD_DIALOG... |
| Language | 0 |
| SubLanguage | 0 |
| Characteristics | 0 |

The one property that needs explaining is the 'Extended' properly.  This property appears for menus and dialogs.  For a menu, it changes the resource from a MENU resource into a MENUEX resource, and for a dialog it changes the resource from a DIALOG resource to a DIALOGEX resource.

# Accelerators Window

The accelerator edit window is accessible by clinking on an accelerator resource in the Resource Window.  It allows you to add keys to an accelerator table.  An accelerator table is a set of keyboard shortcuts which can be used in place of menus or buttons.  Each accelerator has an associated menu identifier, which is usually parsed and passed to the program's main window.  It appears similar to what follows:



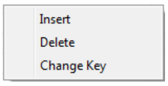
Accelerator - MAINACCELERATORS

| Key | Id | Type | Shift | Alt | Control | Noinvert |
|---|---|---|---|---|---|---|
| VK_ESCAPE | IDM_CLOSEFIND | VIRTKEY | NO | NO | NO | NO |
| VK_F2 | IDM_BOOKMARK | VIRTKEY | NO | YES | NO | NO |
| VK_F2 | IDM_NEXTBOOKMARK | VIRTKEY | NO | NO | NO | NO |
| VK_F2 | IDM_PREVBOOKMARK | VIRTKEY | YES | NO | NO | NO |
| VK_F2 | IDM_BOOKMARKWINDOW | VIRTKEY | NO | NO | YES | NO |
| VK_F3 | IDM_FINDNEXT | VIRTKEY | NO | NO | NO | NO |
| VK_F4 | IDM_TILEHORIZ | VIRTKEY | YES | NO | NO | NO |
| VK_F4 | IDM_TILEVERT | VIRTKEY | NO | NO | YES | NO |
| VK_F5 | IDM_RUN | VIRTKEY | NO | NO | NO | NO |
| VK_F5 | IDM_RUNTO | VIRTKEY | NO | NO | YES | NO |
| VK_F5 | IDM_STOP | VIRTKEY | NO | YES | NO | NO |

Here we have defined CTRL-A, PAGE HOME, CTRL-HOME, and SHIFT-RIGHT. By using the Accelerator Context Menu, rows can be inserted or removed, and the key can be changed by pressing the key sequence to use. And the various qualifiers such as CONTROL, ALT and SHIFT can be modified. Usually, OCIDE will try to set these qualifiers according to the key sequence pressed, however, it may not be able to intercept the ALT key sequences so the ALT modifier must be set by hand.

## Accelerators Context Menu

The accelerators context menu is accessible by right-clicking in the Accelerator Edit Window. It allows inserting and deleting rows. It also has an option that allows changing the key sequence. It is invoked by right- clicking the accelerator edit window. It appears as follows:



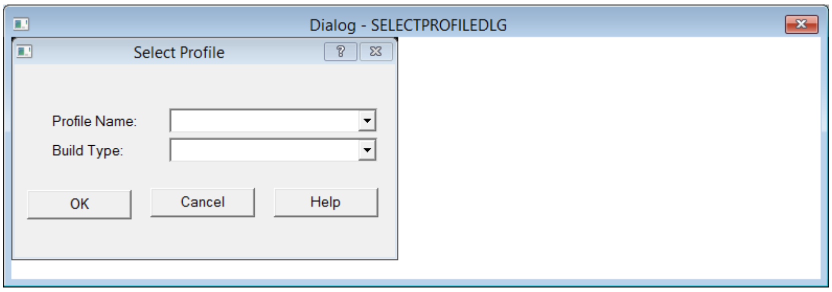
**Insert** inserts a new row

**Delete** removes the selected row

**Change Key** allows changing the key sequence for a row. When this is pressed a small box will appear with the word 'cancel'. You can either press it to cancel the change, or press a key (including shift or control) to set a key sequence for this accelerator.

## Dialog Window

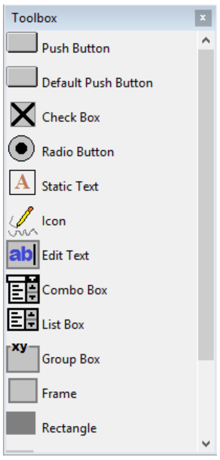
The dialog edit window is accessible by selecting a dialog resource in the Resource Window. It allows editing of dialogs. Typically, a control will be dragged from the Dialog Control Toolbox, or dragged/resized while on the dialog. Extended properties of the control can be set in the Resource Properties window after clicking on the control to select it. Actions taken from either the Resource Menu or the Dialog Context Menu can effect changes to the dialog. The dialog edit window looks similar to what follows when a couple of controls have been dragged on to it.



Here no title has been set for the dialog. There is a combo box with some text to the left, and an edit box with the text to the left. The common buttons 'ok' and 'cancel' have been added off to the right.

## Dialog Control Toolbox

The dialog control toolbox is in the same view as the Properties Window. It shows a list of basic WIN32 controls that may be added to a dialog via the editor. Dragging a dialog from the control toolbox to an active Dialog Edit Window adds a new instance of the control to the dialog.



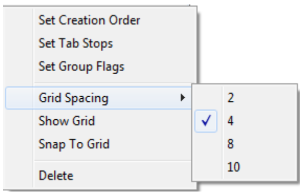
The types of controls OCIDE understands include:

- Push Button
- Default Push Button
- Check Box
- Radio Button
- Static Text
- Icon

- Text Editor
- Combo Box
- List Box
- Group Box
- Frame
- Rectangle
- Etched lines
- Horizontal Scrollbar
- Vertical Scrollbar

## Dialog Context Menu

The dialog context menu is accessible by right-clicking in the Dialog Editor Window. Many of these items are similar to items on the Resource Menu. It appears as follows.



**Set Creation Order** the dialog editor will go into the 'Set Creation Order' mode. In this mode, selecting the controls one by one will set the order the controls will be created. This affects the order for tabbing through the windows, and it will affect visibility if one window overlaps another.

**Set Tab Stops** the dialog editor will go into the 'Set Tab Stops' mode. In this mode, clicking on each control will set it's 'Tab Stop' setting. The 'Tab Stop' setting indicates whether hitting tab on the preceding control will cause this control to get focus. Each control will be outlined in red if it has no tab stop setting, or blue if it does.

**Set Group Flags** the dialog editor will go into the 'Set Group Flags' mode. In this mode, clicking on each control will set it's 'Group Flag' setting. The 'Group Flag' setting indicates whether this control will be in the same group as adjacent controls. The group flag setting is typically used for things like a set of radio buttons, each of which excludes the others when pressed. Each control will be outlined in red if it has no group flag setting, or blue if it does.

**Grid Spacing** this sets the density of the grid, when it is shown

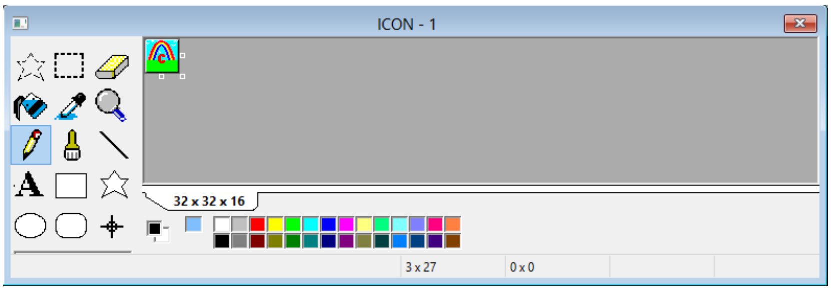
**Show Grid** show a grid to help align controls

**Snap To Grid** when a control is created or moved, make sure its upper right corner aligns with the grid.

**Delete** removes the selected control from the dialog.

## Image window

The image editor window is accessible by selecting an image resource in the Resource Window. This is a similar editor to older paint programs. The image editor window has a context menu similar to the context menu seen on the Resource Menu, which allows some basic functionality. The Image editor window appears as follows:



The image editor window has a command palette to the left, which allows selection of common drawing functions such as selecting parts of an image, or drawing lines or drawing with brushes.

To the right of that is the drawing palette. Here is where images are drawn

Below the image editor command palette is a secondary command palette that allows fine tuning of the command selection, for example selecting the width of a line or the combination of line and fill that will be drawn for a rectangle.

To the right of the secondary palette is a small diagonal color view which shows the colors that are currently selected. The upper left corner shows the color that will be selected by pressing the left mouse button, and the lower right corner shows the color that will be selected by pressing the right mouse button.

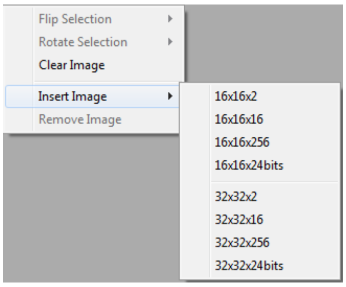
For cursors and icons, there will be a blue box which allows selection of the 'transparent' color. Left or Right clicking on it will make it the selected color for that mouse button. For bitmaps, this is not there.

To the right of the transparent color selection or the diagonal color view is the palette. Left or Right clicking on a palette item will make it the selected color for that mouse button. Also, double clicking on a palette item will bring up the Color Dialog, allowing the color of that palette item to be changed.

For cursors and icons, there will be a small tab bar below the drawing palette. This allows selection of which item from a group of items is being viewed or edited.

## Image Context Menu

The image context menu is accessible by right-clicking on the Image Edit Window. It holds basic functionality for editing the image, similar to what is found on the Resource Menu. It appears as follows:



**Flip Selection**
**Horizontally** flips the selection horizontally

**Vertically** flips the selection vertically

**Rotate Selection**
**90 degrees clockwise** rotates the selection 90 degrees clockwise
**90 degress counterclockwise** rotates the selection 90 degrees counterclockwise
**180 degrees** rotates the selection 180 degrees

**Clear Image** set the image to the background color for bitmaps, or to see through for icons and cursors

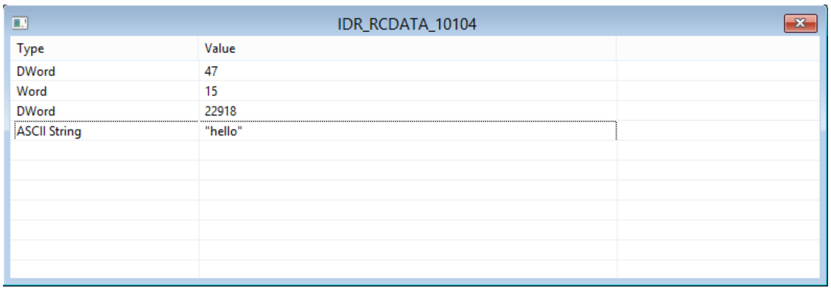
**Clear Selection** set the selection area to the background color for bitmaps, or to the see through color for icons and cursors

**Insert Image** allows an image of a different size or color format to be inserted. In the menu items, the first number is width, the second number is height, and the third number is the number of colors it can have. So 32x32x16 is 32 wide, 32 high, and 16 colors.

**Remove Image** removes one the selected image. However, there must always be at least one image.

## RCData Window

The RCData edit window is accessible by selecting an RCData resource from the Resource Window. It appears similar to what follows:



IDR_RCDATA_10104

| Type | Value |
|---|---|
| DWord | 47 |
| Word | 15 |
| DWord | 22918 |
| ASCII String | "hello" |

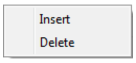
This window allows adding arbitrary data to a resource. The types of data is selected in the left column, and the data is presented in the right column. Valid data types are:

- DWORD (4 bytes)
- WORD (2 bytes)
- ASCII text

At present the RCData window will not allow editing Unicode text.

## RCData Context Menu

The RCData context menu is accessible from the RCData Edit Window. It allows for inserting and removing rows. It appears similar to what follows:

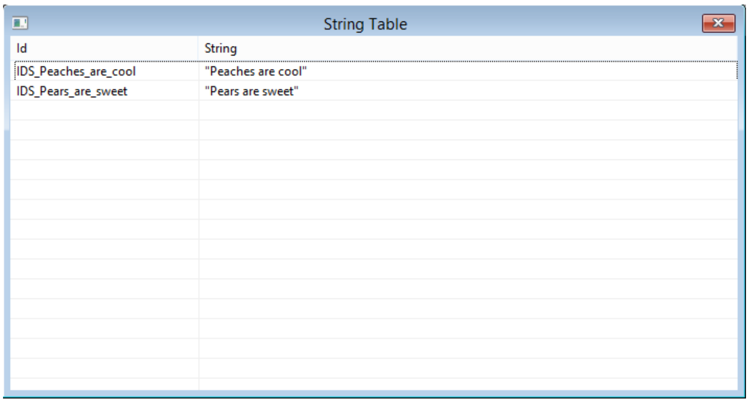


**Insert** inserts a new row

**Delete** removes the selected row

## String Table Window

The String Table edit window is accessible by selecting a String Table resource from the Resource Window. It has a related context menu that allows inserting and deleting new strings. It appears similar to what follows:



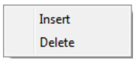
String Table

| Id | String |
|---|---|
| IDS_Peaches_are_cool | "Peaches are cool" |
| IDS_Pears_are_sweet | "Pears are sweet" |

Here the left-hand column shows the resource identifier of the string, and the right-hand column shows the string.

## String Table Context Menu

The String Table context menu is accessible from the String Table Edit Window. It has menu items for inserting and removing rows. It appears similar to what follows:

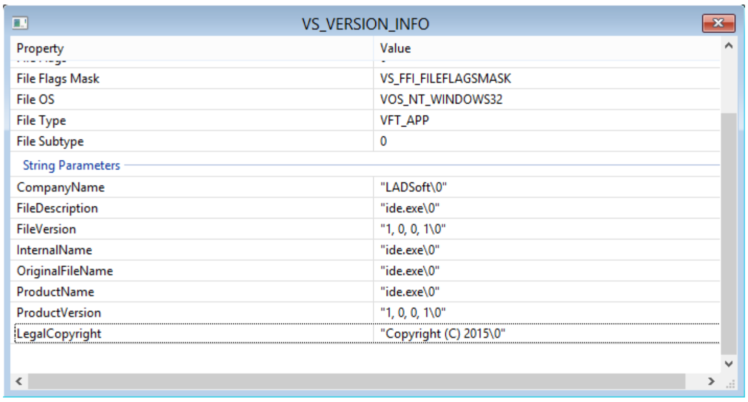


**Insert** inserts a new row

**Delete** removes the selected row

# Version Data Window

The Version Data Window is accessible by selecting the Version resource from the Resource Window. It allows editing of version information for the program or DLL. This information can often be displayed by the operating system. It has a related context window for inserting or removing string-related version information. It appears as follows:



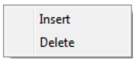
VS_VERSION_INFO

| Property | Value |
|---|---|
| File Flags Mask | VS_FFI_FILEFLAGSMASK |
| File OS | VOS_NT_WINDOWS32 |
| File Type | VFT_APP |
| File Subtype | 0 |
| String Parameters | |
| CompanyName | "LADSoft\0" |
| FileDescription | "ide.exe\0" |
| FileVersion | "1, 0, 0, 1\0" |
| InternalName | "ide.exe\0" |
| OriginalFileName | "ide.exe\0" |
| ProductName | "ide.exe\0" |
| ProductVersion | "1, 0, 0, 1\0" |
| LegalCopyright | "Copyright (C) 2015\0" |

Note that unlike other editor windows, this window comes pre-populated and typically it will just be a matter of changing the information to match your program.

# Version Data Context Menu

The Version Data context menu is accessible by right-clicking on the Version Data Edit Window. It allows inserting and deleting string data. It appears similar to what follows:

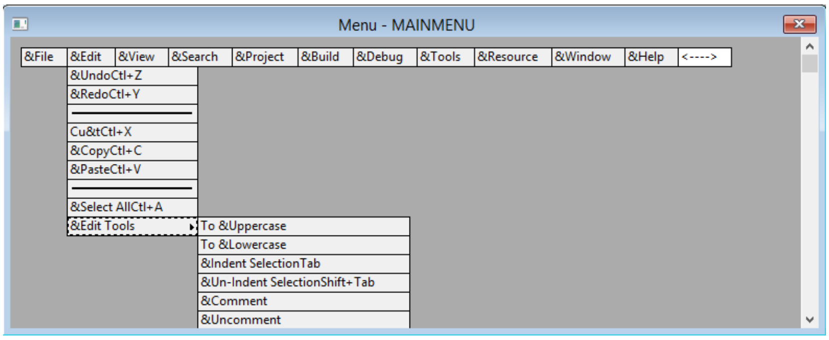


**Insert** inserts a new row. This will only insert rows in the bottom or string section.

**Delete** removes the selected row. This will only remove rows from the bottom or string section; the other information rows are fixed.

# Menu Window

The Menu Window is accessible by selecting a Menu resource from the Resource Window. It allows editing menus. It has an associated context menu, which is used for inserting or deleting menu items. It appears similar to what follows:



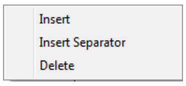
Here, the WORKAREA menu item has been clicked to drop down the WORKAREA menu. Additionally, Close Work Area has been selected, as can be seen by the white box to the right of it.

The white boxes with arrows are the places where menu items can be inserted. If a menu item is inserted to the right of another menu item it becomes an item on a sub menu. The exception is the top row, where items to the right become new main menu selectors.

A menu item may also be inserted at the bottom of a menu, or in front of an existing menu item. Additionally, a SEPARATOR may be inserted for grouping menu items.

# Menu Context Menu

The Menu Context Menu is accessible by right-clicking on a menu item in the Menu Editor Window. It allows inserting and removing menu items. It appears similar to what follows:



**Insert** inserts a new menu item

**Insert Separator** inserts a new separator

**Delete** removes a menu item