Submit New Art

Debugging Intel(R) Array Building Blocks Programs in Microsoft* Visual Studio*

Introduction :

This article explains debugger integration for Intel(R) ArBB in a Microsoft* Visual C++* IDE. It also demonstr the basic usage of the debugging facility.

Version :

Intel® Array Building Blocks 1.0 Beta 1 for Windows* OS

Application Notes :

The debugger integration allows programmers to inspect the content of Intel® Array Building Blocks container using Microsoft* Visual C++* debugger. The Intel(R) ArBB installation process on Windows* OS does not inst this feature automatically. Users have to manually add this feature following the instructions in this article.

Prerequisites :

Intel® Array Building Blocks 1.0 Beta 1 product must be installed. For information on how to get and install Intel(R) ArBB software on Windows* platform, refer to this KB article.

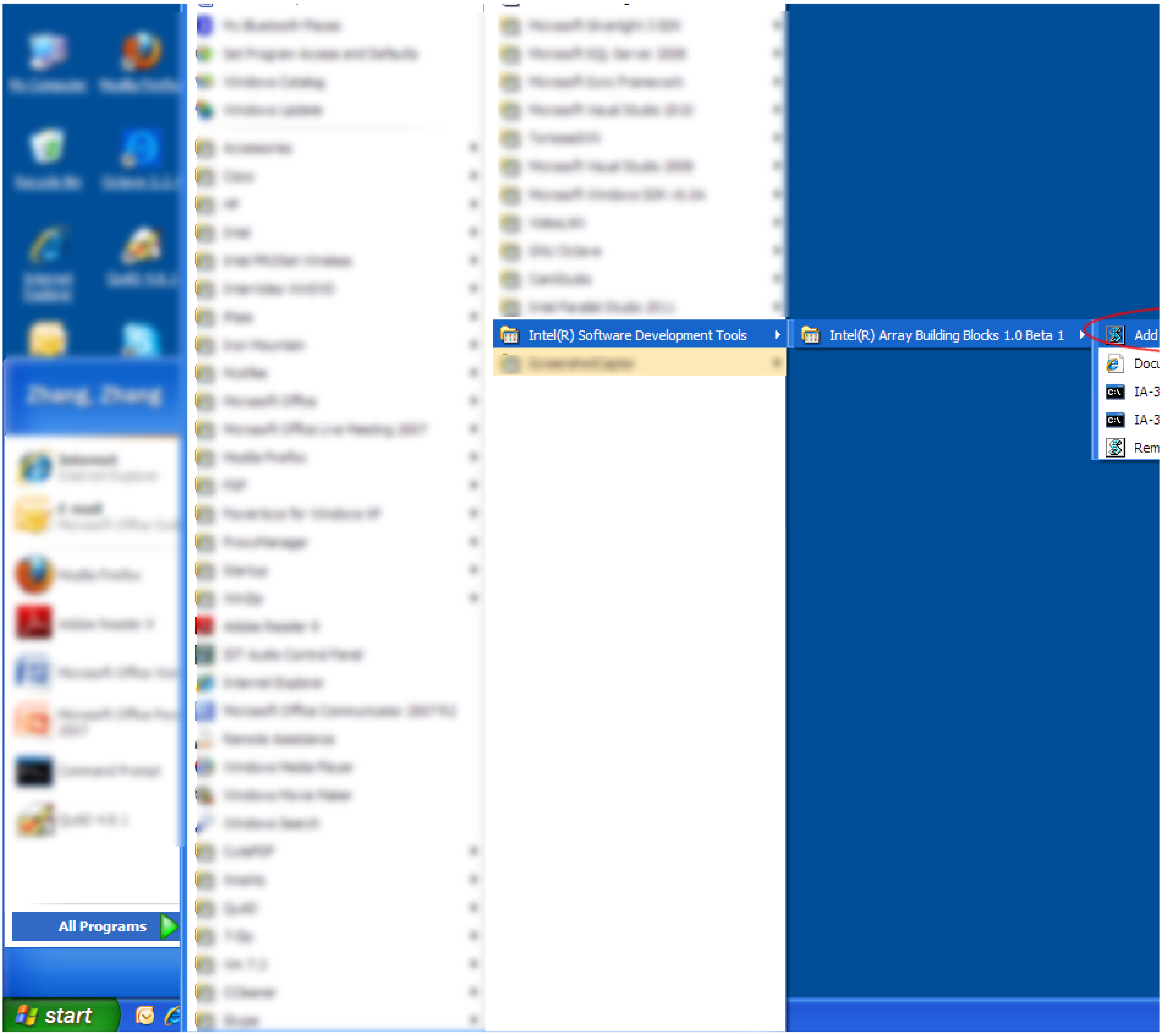
One of the following supported Microsoft* Visual C++* IDE products must be installed:

- Microsoft* Visual C++* 2005 SP1
- Microsoft* Visual C++* 2008

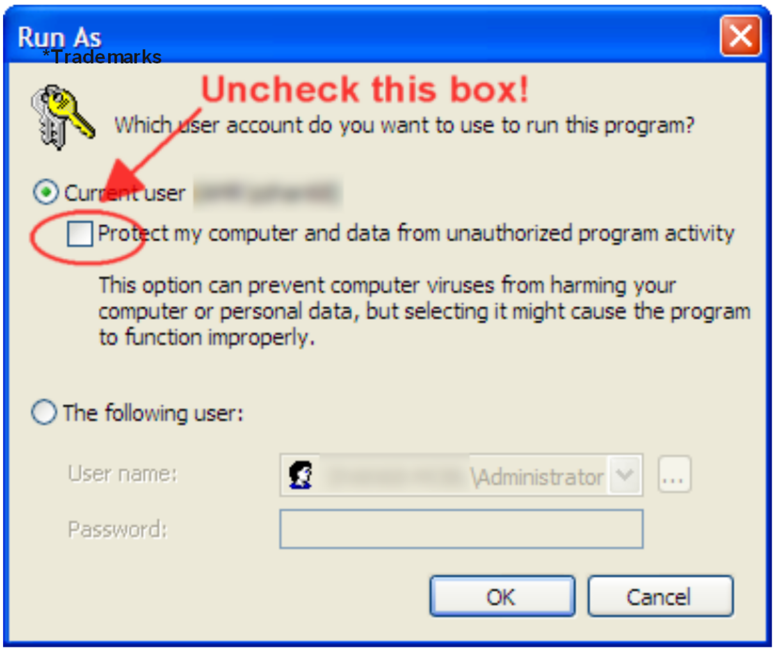
- Microsoft* Visual C++* 2010

Add the Debugger Integration:

To enable this integration, go to **start -> All Programs -> Intel(R) Software Development Tools -> Intel(R) Array Building Blocks 1.0 Beta 1**, then click **Add MSVC Debugger Integration**, as shown in the picture bel


start
All Programs
Intel(R) Software Development Tools
Intel(R) Array Building Blocks 1.0 Beta 1
Add
Docu
IA-3
IA-3
Rem

This will run a Windows* script that needs to modify a restricted area (the *Program Files* folder). So you may prompted a **Run As** dialog box as shown below. Be sure to uncheck the system protection box before click **OK** Otherwise, the installation will fail.


Run As

*Trademarks

Uncheck this box!

Which user account do you want to use to run this program?

● Current user [redacted]

☐ Protect my computer and data from unauthorized program activity

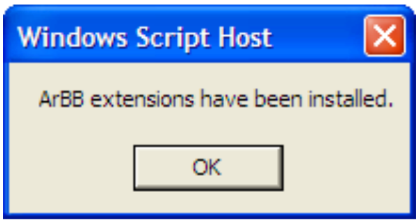
This option can prevent computer viruses from harming your computer or personal data, but selecting it might cause the program to function improperly.

○ The following user:

User name: [redacted]\Administrator

Password:

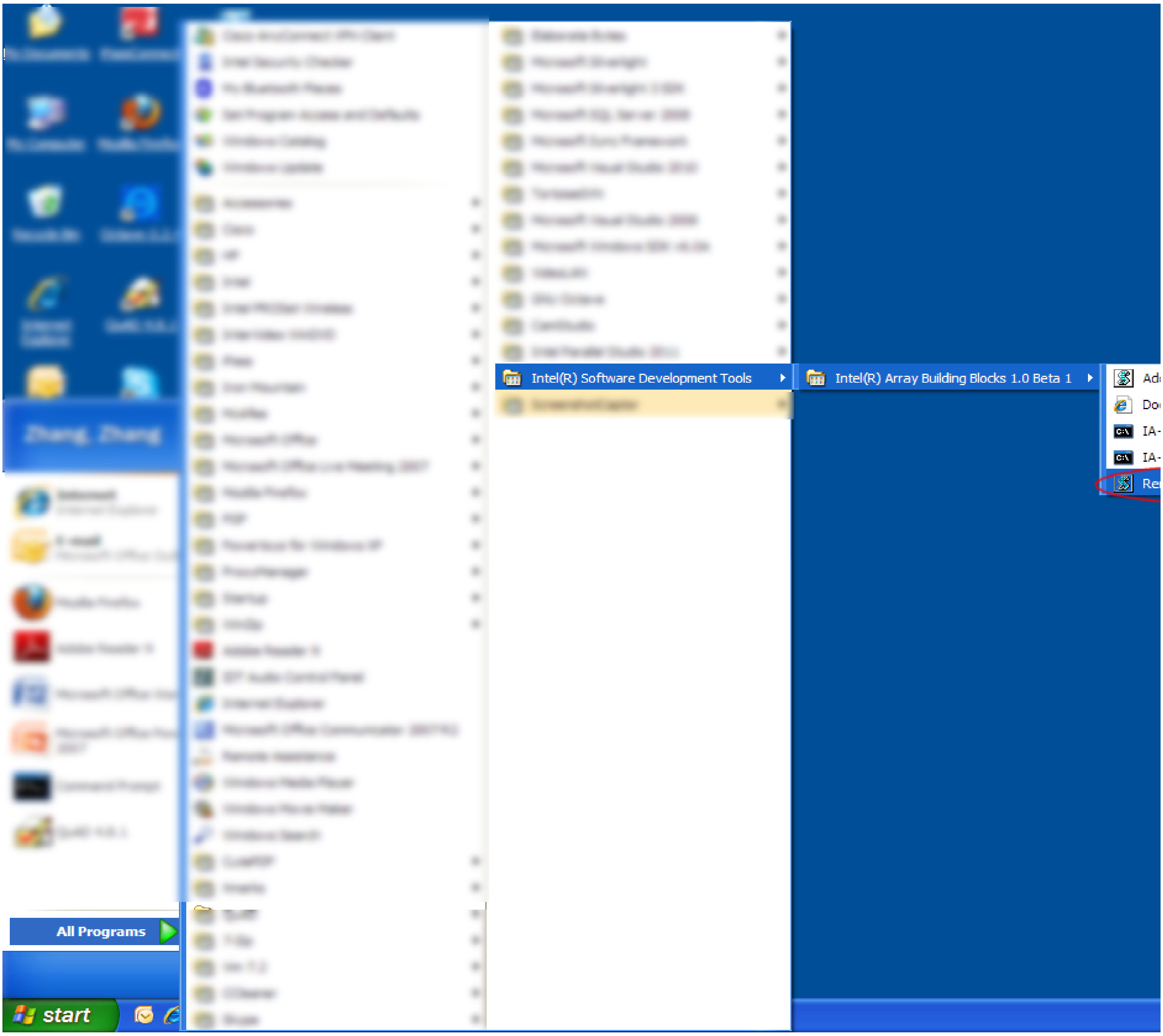
If the installation is successful, you will see a confirmation like this:


Windows Script Host

ArBB extensions have been installed.

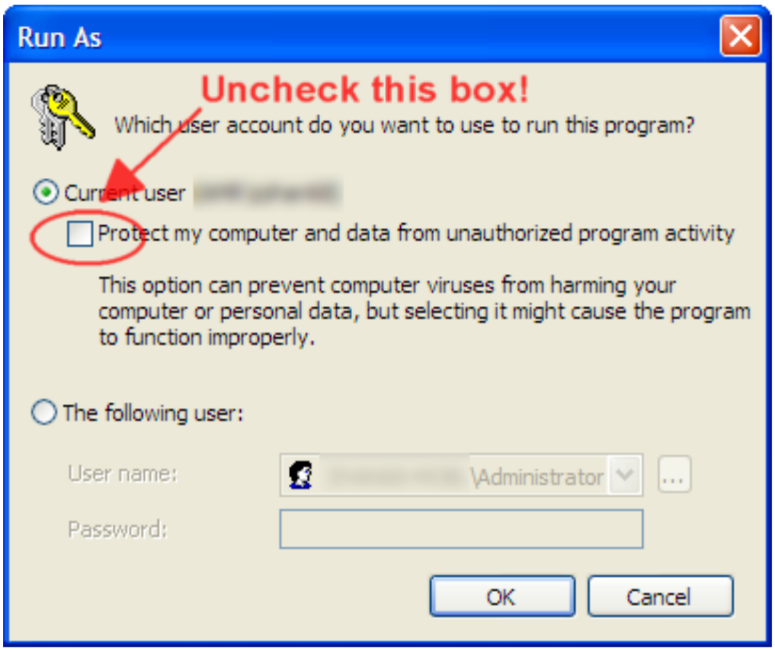
Remove the Debugger Integration:

To disable the integration, click **Remove MSVC Debugger Integration** from the same menu, as shown in the picture below:



Intel(R) Software Development Tools
Intel(R) Array Building Blocks 1.0 Beta 1
Ad
Do
IA-
IA-
Re
All Programs
start

Again, you may be prompted a **Run As** dialog box as shown below. Be sure to uncheck the system protection b
before click **OK**. Otherwise the uninstallation will fail.



Which user account do you want to use to run this program?

● Current user [redacted]

☐ Protect my computer and data from unauthorized program activity

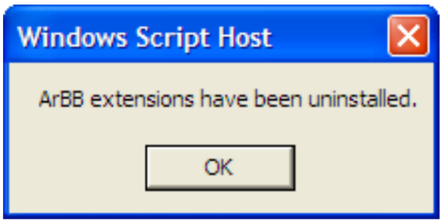
This option can prevent computer viruses from harming your computer or personal data, but selecting it might cause the program to function improperly.

○ The following user:

User name: [redacted]\Administrator

Password:

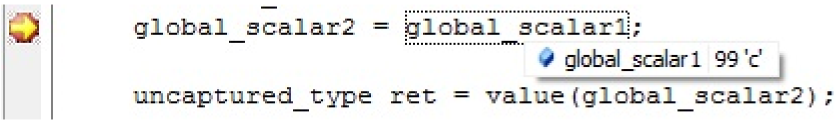
If the uninstallation is successful, you will see a confirmation like this:



ArBB extensions have been uninstalled.

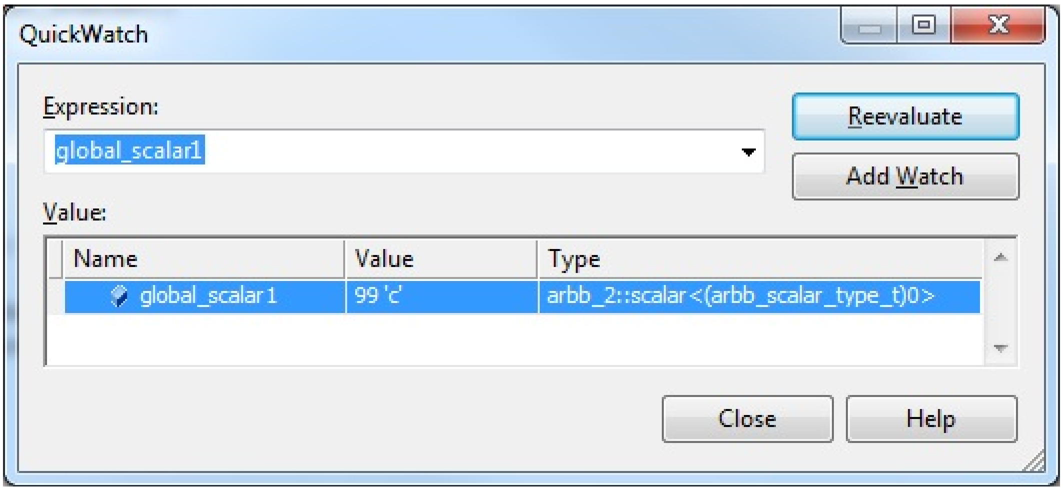
Basic Usage:

- Inspecting Intel® ArBB scalar variables: Scalars can be viewed in several ways. The following examples assume "*global_scalar*" is an **i8** type variable.
  1. The user can hold the cursor over a scalar variable until a "SmartTag" appears, as shown in this pic

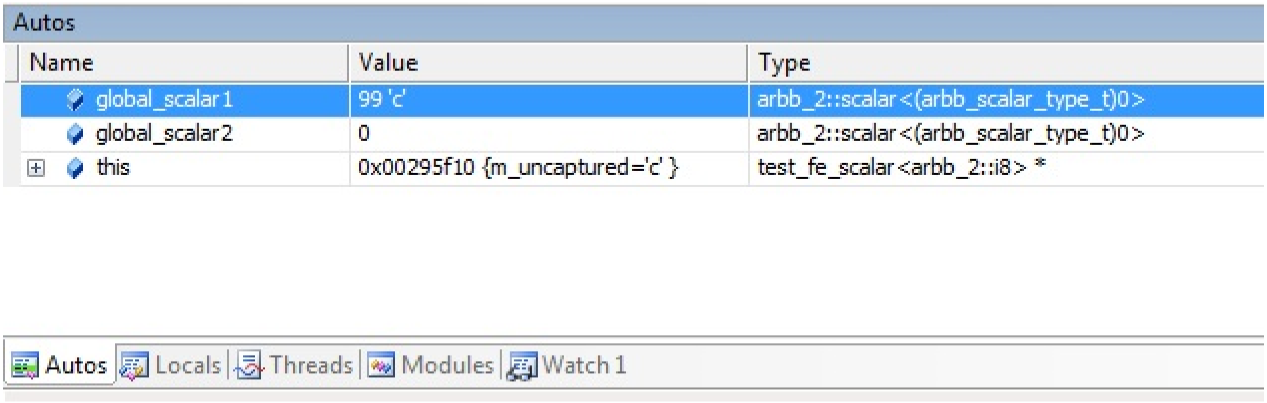
     ```
     global_scalar2 = global_scalar1;
                              global_scalar1 99 'c'
     uncaptured_type ret = value(global_scalar2);
     ```

  2. The user can right-click on the variable and select "Quick Watch" from the pull-down menu, as sh in this picture



Name	Value	Type
global_scalar1	99 'c'	arbb_2::scalar<(arbb_scalar_type_t)0>

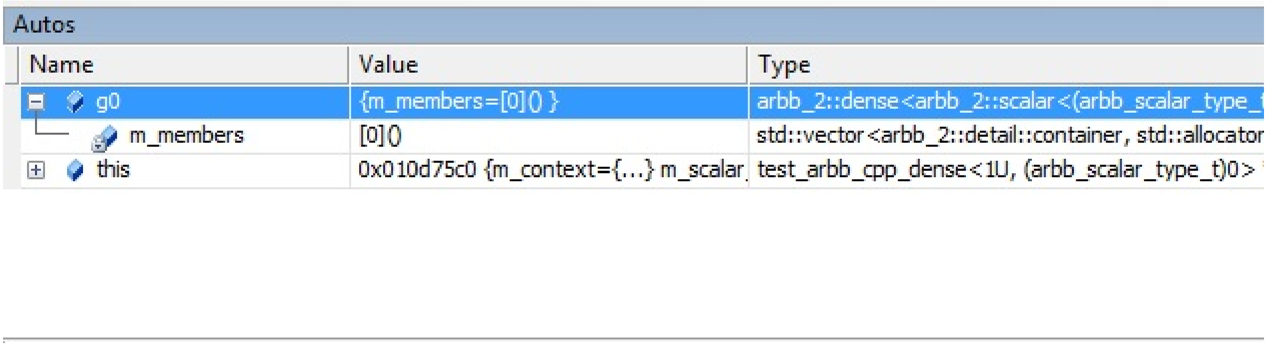
3. The variable can also be displayed in the "Autos", "Locals" or "Watch" debug windows, as shown this picture



Name	Value	Type
global_scalar1	99 'c'	arbb_2::scalar<(arbb_scalar_type_t)0>
global_scalar2	0	arbb_2::scalar<(arbb_scalar_type_t)0>
this	0x00295f10 {m_uncaptured='c' }	test_fe_scalar<arbb_2::i8> *

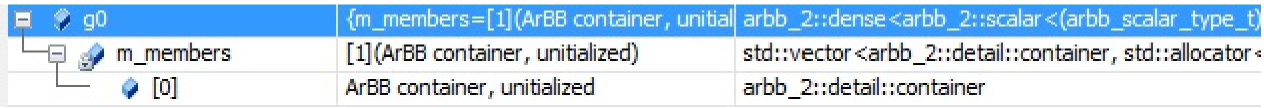
- Inspecting Intel® ArBB containers: Certain high-level information (e.g. the length) of a container, as wel the individual members of the container, can be displayed. The following examples assume "*g0*" is an 1D dense container of **i64** type, and "*g2*" is a 2D dense container of **i64** type.

  1. Before *g0* is initialized, it looks like this

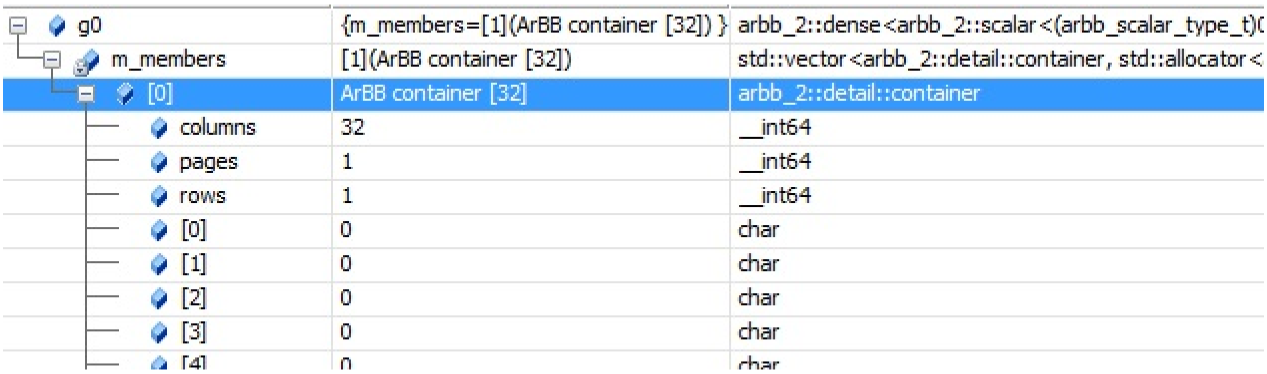
Name	Value	Type
g0	{m_members=[0]() }	arbb_2::dense<arbb_2::scalar<(arbb_scalar_type_
m_members	[0]()	std::vector<arbb_2::detail::container, std::allocator
this	0x010d75c0 {m_context={...} m_scalar	test_arbb_cpp_dense<1U, (arbb_scalar_type_t)0>

  2. After it has been constructed, but before it contains any data, it looks like this

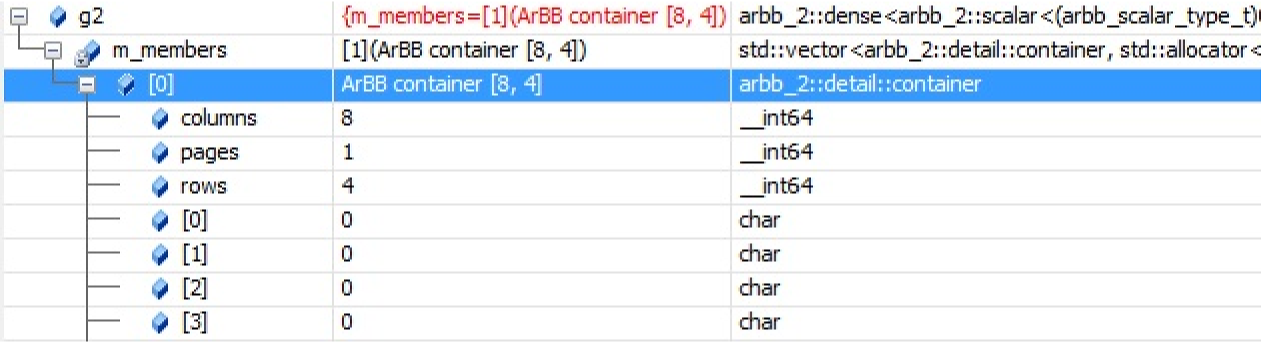
g0	{m_members=[1](ArBB container, unitial	arbb_2::dense<arbb_2::scalar<(arbb_scalar_type_t)
m_members	[1](ArBB container, unitialized)	std::vector<arbb_2::detail::container, std::allocator
[0]	ArBB container, unitialized	arbb_2::detail::container

3. After it has been filled with data, it looks like this



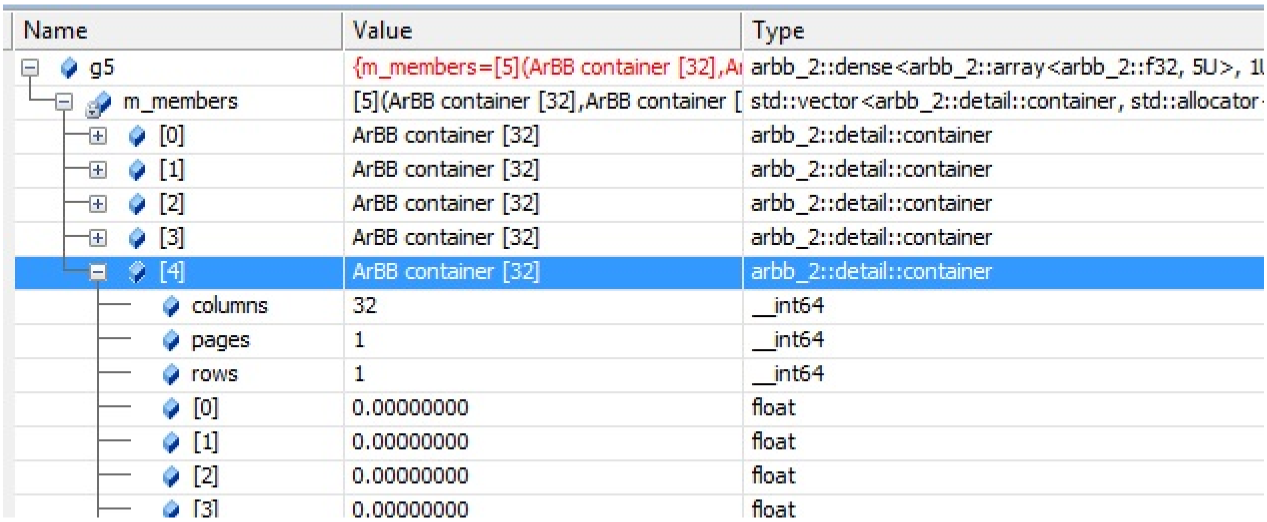
g0	{m_members=[1](ArBB container [32]) }	arbb_2::dense<arbb_2::scalar<(arbb_scalar_type_t)(
m_members	[1](ArBB container [32])	std::vector<arbb_2::detail::container, std::allocator<
[0]	ArBB container [32]	arbb_2::detail::container
columns	32	__int64
pages	1	__int64
rows	1	__int64
[0]	0	char
[1]	0	char
[2]	0	char
[3]	0	char
[4]	0	char

4. For a 2D or 3D dense container, the size of each dimension is shown in the "columns", "rows" or "pages" values. The individual members of the container are shown as a flat array as if the contain flattened in the row-column-page order. This may be improved in future releases. The picture belo illustrates how a 2D dense container (*g2*) is displayed:



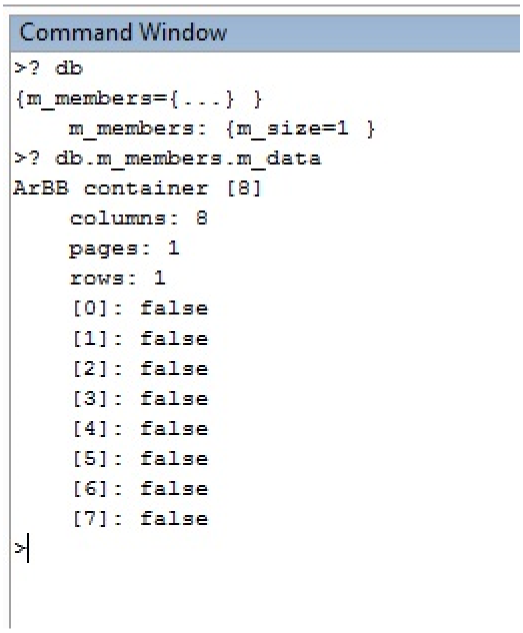
g2	{m_members=[1](ArBB container [8, 4])	arbb_2::dense<arbb_2::scalar<(arbb_scalar_type_t)(
m_members	[1](ArBB container [8, 4])	std::vector<arbb_2::detail::container, std::allocator<
[0]	ArBB container [8, 4]	arbb_2::detail::container
columns	8	__int64
pages	1	__int64
rows	4	__int64
[0]	0	char
[1]	0	char
[2]	0	char
[3]	0	char

5. A nice feature of the debugging support is that it can be used to show the *AoS-to-SoA* conversions Intel® ArBB performs on the containers of structured types. The following example shows that a 1 dense container (*g5*) of length 32. Each of its element is a 5-field struct type. Each field is an **i64** t integer. As we can expect from the *AoS-to-SoA* conversion, fields of the struct type are scattered i different containers. Here we see the original dense container is split into 5 containers, each contai values corresponding to one of the five fields of the original struct type.



Name	Value	Type
g5	{m_members=[5](ArBB container [32],Ar	arbb_2::dense<arbb_2::array<arbb_2::f32, 5U>, 1U
m_members	[5](ArBB container [32],ArBB container [	std::vector<arbb_2::detail::container, std::allocator
[0]	ArBB container [32]	arbb_2::detail::container
[1]	ArBB container [32]	arbb_2::detail::container
[2]	ArBB container [32]	arbb_2::detail::container
[3]	ArBB container [32]	arbb_2::detail::container
[4]	ArBB container [32]	arbb_2::detail::container
columns	32	__int64
pages	1	__int64
rows	1	__int64
[0]	0.00000000	float
[1]	0.00000000	float
[2]	0.00000000	float
[3]	0.00000000	float

6. It is also possible to print Intel® ArBB variables in the Visual Studio* "Command" and "Immediate windows. However, in the case of dense containers, you have to specify a couple of member varial to get to the container level. For instance, if there is a dense container, *db*, containing **booleans**, yo have to specify *db.m_members.m_data* to show the content of the container:



```
>? db
{m_members={...} }
    m_members: {m_size=1 }
>? db.m_members.m_data
ArBB container [8]
    columns: 8
    pages: 1
    rows: 1
    [0]: false
    [1]: false
    [2]: false
    [3]: false
    [4]: false
    [5]: false
    [6]: false
    [7]: false
>
```

Known Issues or Limitations:

1. The debugger can only be used to inspect Intel® ArBB scalars and containers whose elements are of buil types, such as **i32** and **f64**. It does not work well with containers whose elements are of user-defined type
2. The debugger only works for the *emulation mode*. That is, the Intel® ArBB optimization level must be se **O0** using the environment variable **ARBB_OPT_LEVEL**. Programs with big input size may run very slo even crash in this mode.

Do you need more help?
Click tags links for related articles Search Knowledge Base Visit User Forums Get other Support options