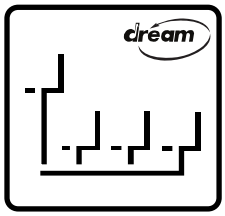


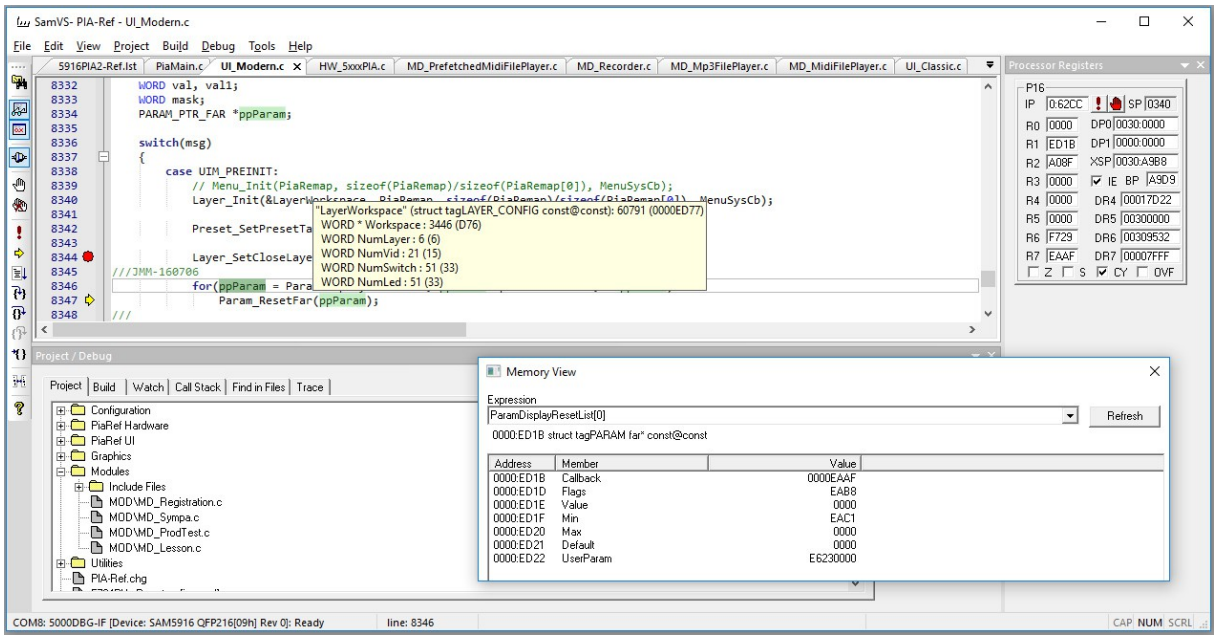
SamVS-C V5.00
Release Notes

List of Contents

SamVS-C – Integrated Development System
Version 5.00 - Rev: 07/10/2019 - © DREAM S.A.S.

1 Release Notes to SamVS V5.00

SamVS V5.00 comes in a completely renovated user interface, this document describes briefly what had changed and what new features have arrived. While this is a significant change, the overall use and workflow did not change. The experienced SamVS user should be able to continue business without reading this document, at the potential cost to miss some new features.



The most obvious changes are:

- Renovated user interface supporting display themes and document tabs.
- Using Scintilla code editor with customizable syntax highlighting, code folding, inlined compiler messages, brace matching, word highlighting and many more.
- Hierarchical project layout.
- Memory View (aka “QuickView”) dialog displays structure contents and allows edits.
- Expression hovering provides more details.

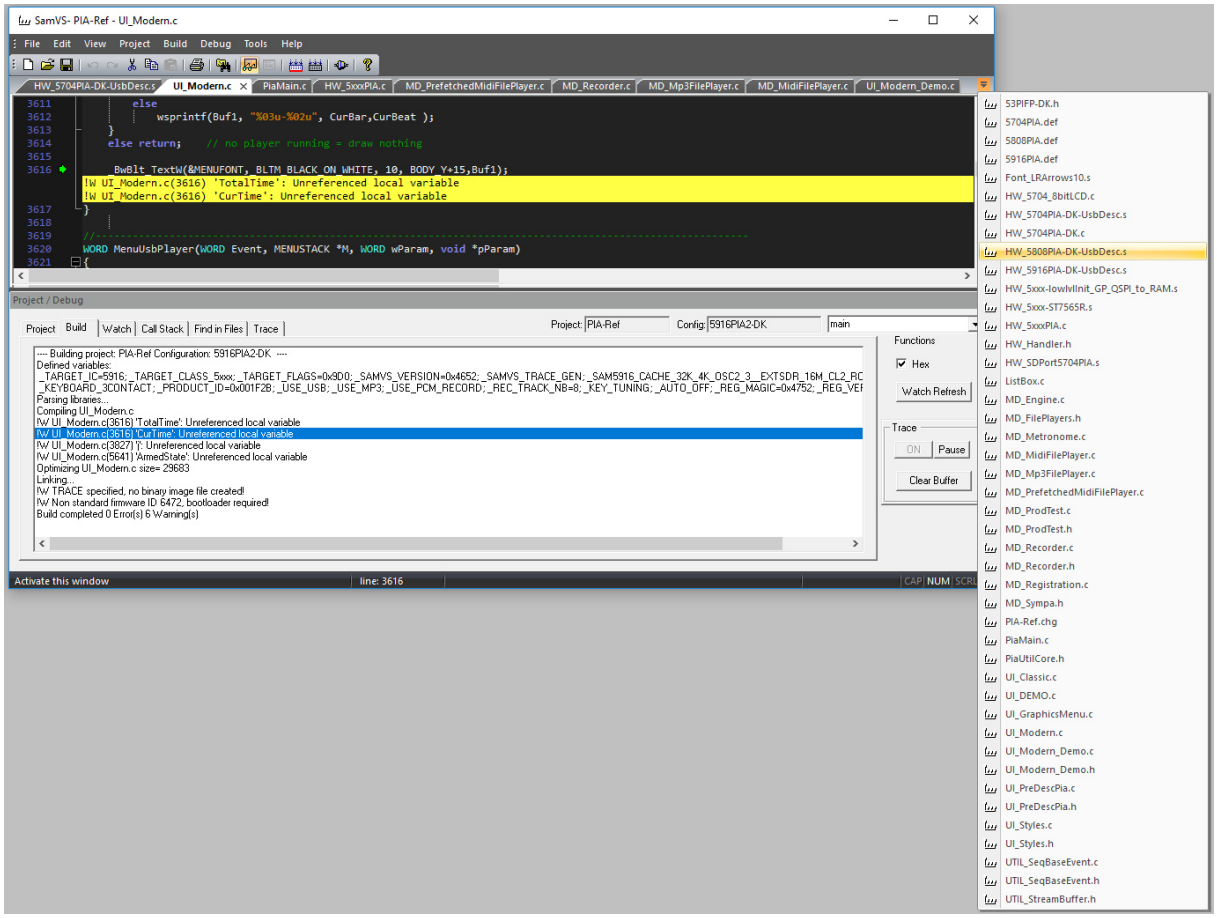
This release includes several fixes and enhancements as well:

- Watches properly expand structures and pointers, even from pointers in registers.
- Memory View allows to edit values (like the watches do) and no longer requires a warning for large dumps.
- Search phrases now kept in least recently used (LRU) lists.
- Syntax highlight also for assembler.
- P16 breakpoints survive text edits.
- Code editor handles large files much better.
- P24 debugging supports modification of P24 registers and preserves breakpoints which are automatically set whenever SamVS detects a reload of P24 program.
- Various performance enhancements.

The following pages describe those changes in much more detail.

2 New User Interface

The most significant change in user interface is obviously the new tab navigation for documents on top of the edit area:



This effectively eliminates the former Window menu along with the window selection dialog. If you have more documents open than tabs fit into the window, you'll have the alphabetically sorted window list available on the button at the top right.

Along with the mouse navigation using the tabs or the list, you may still sweep through the open documents using Strg+Tab resp. Shift+Strg+Tab key combinations in least recently used order.

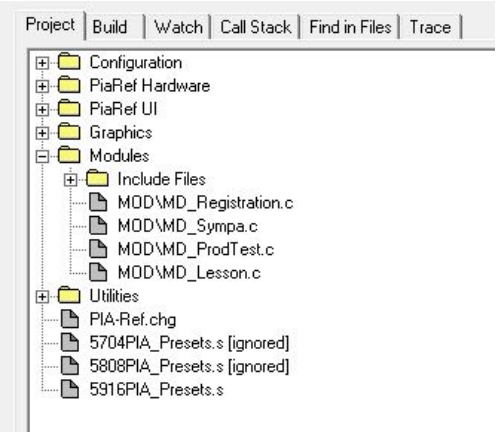
You'll notice that the document tabs do intentially not update with the LRU order, so they seem to jump around when tabbing through the documents. However, if you save and load a project, tabs are restored from the LRU list.

Another obvious change can be found in the View menu, where you now have the option to select between various color themes for the overall layout and different editor color schemes to adjust SamVS to match your personal preferences.

And finally the dockable bars now resize with the window frame and may dock in more combinations than before.

3 Project Tree

If you need to manage larger projects, the new project management may be pretty helpful. Instead of holding all project files in a linear list, you now have the option to organize the project's sources in a hierarchical tree, as you see in the screenshot.



This is a logical structure only and does neither affect the physical location of your source files nor the build process.

SamVS does not expect any specific structure, so you may build any hierarchy which best fits to your project.

To add such a project folder simply use Add Project Folder from the context menu and give it any name. Then move some files onto that folder using drag&drop.

You'll notice that the first file drops into the folder but the second will not until you open the folder clicking on the plus sign. This is intentional behavior since you may want to sort something behind a folder (when it is closed) and an empty folder can not be opened. So when dropping an item onto an empty folder it will be placed inside.

Of course you may place folders inside folders, as you can see from the screenshot and move around a complete subtree to a different location.

Deleting a folder is possible from the context menu as well (or pressing DEL on a folder). This operation only "pulls" the folder from the hierarchy so all contained nodes will occur on the previous folder location.

File locations and open status will be preserved in your project, but folders are always sorted on top. The previous sort functions from the context menu are no longer necessary and are no longer available.

Toggling the ignored state is still possible from the context menu, but the tree view does no longer allow to select multiple items. So toggling multiple entries may become a tedious task. In case you frequently need to toggle same elements you may place them into a project folder and then toggle the folder instead.

As a quick alternative you may use the keyboard for tree navigation and manipulation. Left and right cursor keys close resp. open a tree node, up and down move, well, up and down. Pressing # will toggle ignore state and DEL may remove an item from the project.

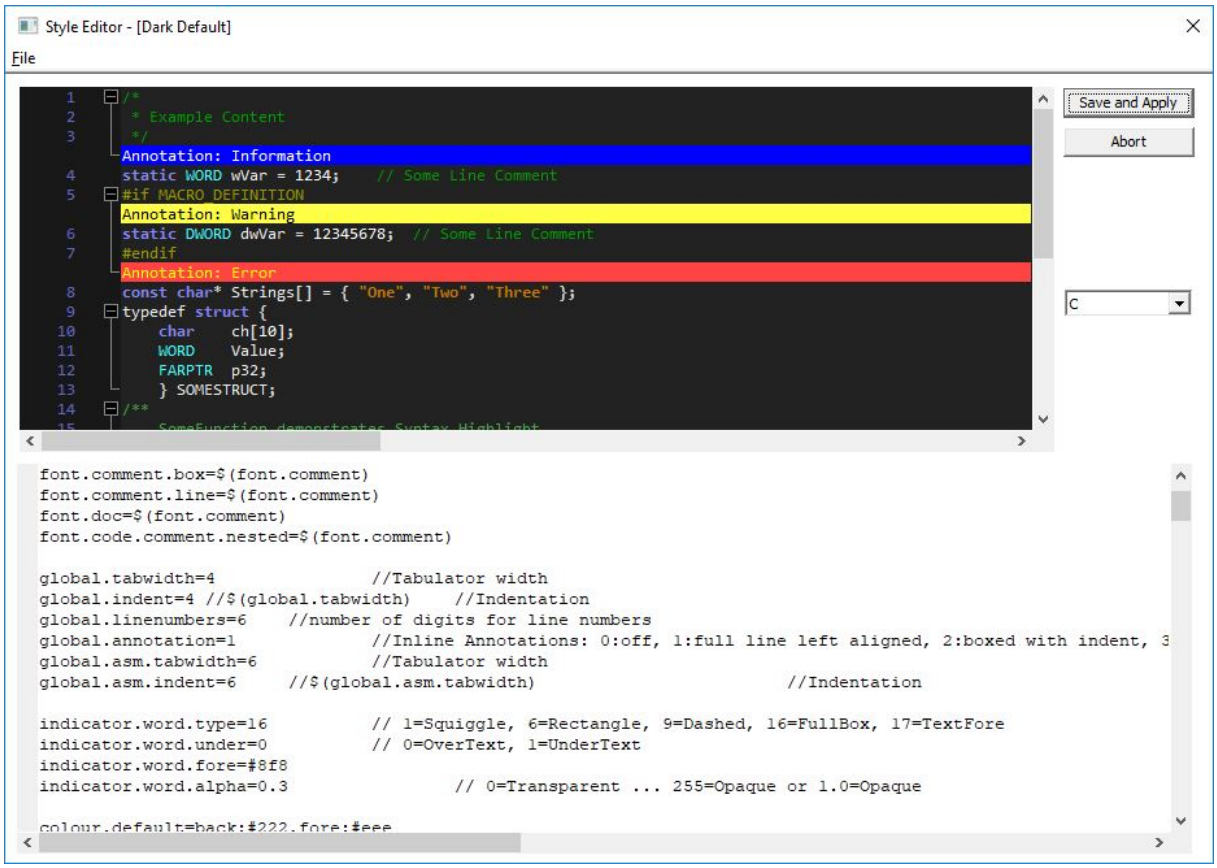
4 Scintilla Code Editor

When it comes to code editing, the new Scintilla (www.scintilla.org) code editor is the most significant enhancement. It not only provides highly efficient and configurable syntax highlighting, but also features some quite useful other functions:

- Code Folding
  Scintilla understands the document structure and allows to collapse parts of your document for better visibility. By default blocks are considered to begin and end at curly braces or preprocessor conditionals. For assembler sourcefiles any XFUNC/LFUNC is considered a code block which can be folded away.
- Line Numbers
  Sometimes it may be helpful to have line numbers next to the code, its rather easy to have them with Scintilla, or turn it off, if not needed.
- Brace matching
  Ever wondered if braces on more complex statements match? Scintilla automatically highlights matching braces or display them red if unmatched.
- Word highlighting
  Sometimes it may be handy to highlight occurrences of the same word in your document, to see where a variable is used. Just double click on a word for this.
- Block operations
  Of course Scintilla supports block operations. You may select an area (instead of a couple of lines) or just a column while holding the ALT key. You may then cut/copy and paste such a block or just press TAB to indent.
- Inline annotations
  Scintilla allows to insert annotations to code lines which is used to duplicate build errors and warnings right at the text location.
- Find and Replace using regular expressions
  Similar to the find in files functionality, you now may also use regular expressions in document-wide find and replace operations. Another nice feature is to limit a replace operation to only a selected portion of text.
- Undo and Redo over save points
  Undo history no longer cleared when saving a file. As long as the documents stays open in a SamVS session, you may undo text edits even after debugging.

4.1 Scintilla Highlight Editor

The configuration possibilities of Scintilla are too complex to fit into a comfortable user interface without loosing flexibility. So SamVS provides an interactive configuration editor to modify existing or even build your own styles:



Its probably out of scope to explain all possibilities, but the included templates should contain enough comments to find your way through the massive amount of options.

You'll find the editor in the View→Syntax Highlight menu, it will load with the currently selected style. The upper editor is an editable preview of some text (you may switch language from the dropdown on the right) and the lower contains the configuration properties (to match the semantic from Scintilla.org).

Properties are basically hierarchical, so something assigned to a property "font" will affect any font, something assigned to "font.comment" will only affect fonts used in comments. Properties may contain a wildcard as well, so "style.*.34" defines brace highlighting for any language.

Please note that various highlight styles are defined by numbers. Check out the comments in the predefined styles whats supported.

Various aspects may be assigned to such properties:

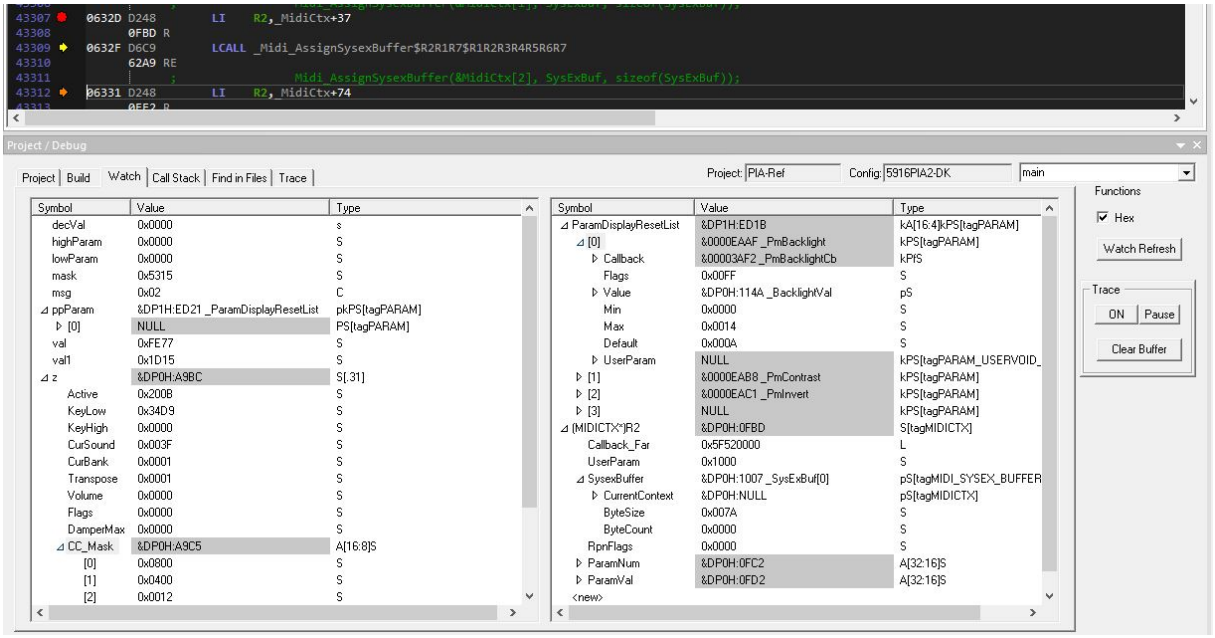
- font:<fontTypeface>
  typeface is the actual name of the font like “Arial” or “Consolas”
- size:<fontPixelHeight>
  the size of a font in pixels. Yes, you are free to specify different fonts and sizes for any aspect.
- fore:<colourForeground>
  Colour used for the text.
- back:<colourBackground>
  Colour used for the background of the text.
- <number>
  Just a number, use depends on the property.
- $(<property>)
  Refers to another property. This brings in the power for controlling many properties from just a few.
- <wordlist>
  Wordlists are basically space separated identifiers, which are used for keyword highlighting. You’ll find examples at the end of the provided styles. If you need a word which does not look like an identifier (i.e. contains symbols, spaces or similar), enclose it in double quotes.

When editing the style code, you’ll get an instant feedback in either the preview window or, in case of errors, as annotated lines in the code editor.

If you are happy with your modifications, just press Save and Apply (or use save from the file menu). Your personal styles need to go into the config folder next to bin, where SamVS.exe resides. The save dialog takes care of that.

5 Debugging

Another big improvement can be seen when you start a debugging session. The watch windows now not only follow structured data through levels and pointers, you may use typecasts as well to interpret any address or literal data as a pointer to something special. This does work with registers as well, as you may see here:



The snippet stops in the listfile where register R2 is loaded with a pointer to a MIDICTX structure. The lower expresseion in the right watch windows displays the content of that structure, the expression entered is: (MIDICTX*)R2

If possible, the SamVS debugger resolves symbol names from addresses and can deal with far pointers as well. Check out ppParam in the left watches or the unfold ParamDisplayResetList in the right one. The third column displays native type information, so you can verify to what type your expression actually evaluates.

The legacy hex checkbox is still available and provides a quick change from hexadecimal to unsigned decimal as a default for all values (except pointers, which are always displayed hex).

If you want to display individual fields differently, you may either pick a format from the context menu or add a format specifier to your expression.

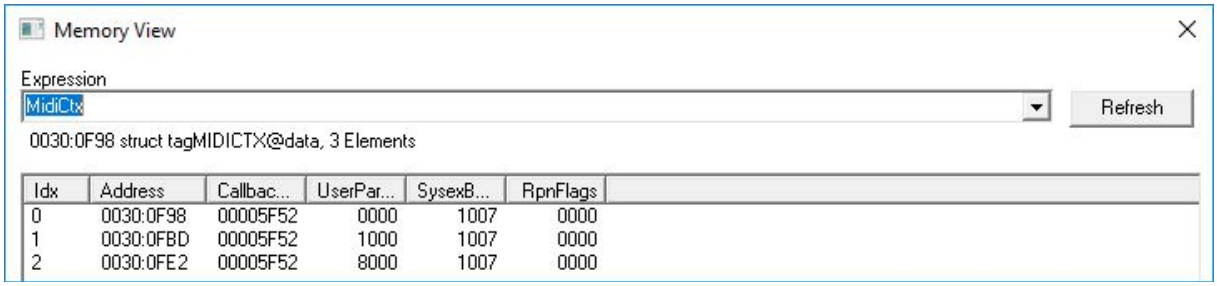
A format specifier starts with a # symbol and is followed by an optional repeat count and an optional format type, which is one of:

- x – hexadecimal
- d or s – signed short
- u – unsigned short
- f – 32 Bit float

The repeat count may be placed for any expression which evolves to an address and enforces the watch to list the requested number of elements, regardless if the expression actually refers to an array or not.

6 Memory View

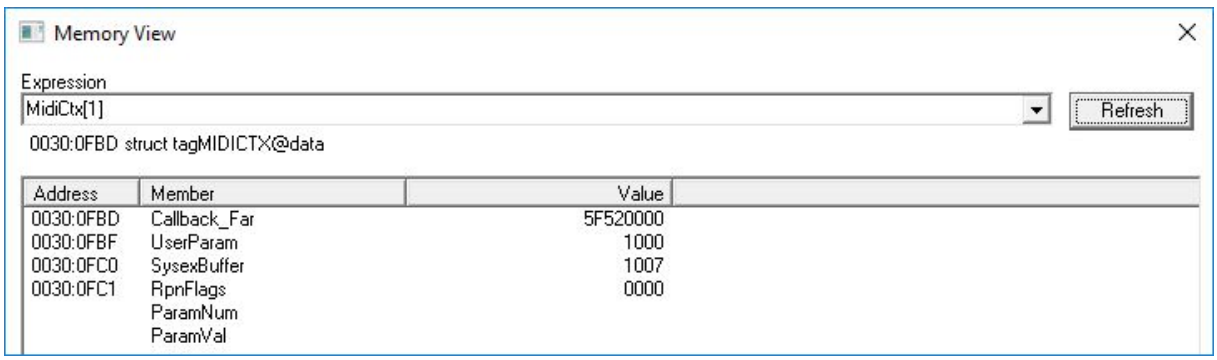
The probably most significant change from previous habits will introduce the Memory View dialog, formerly known as QuickView:



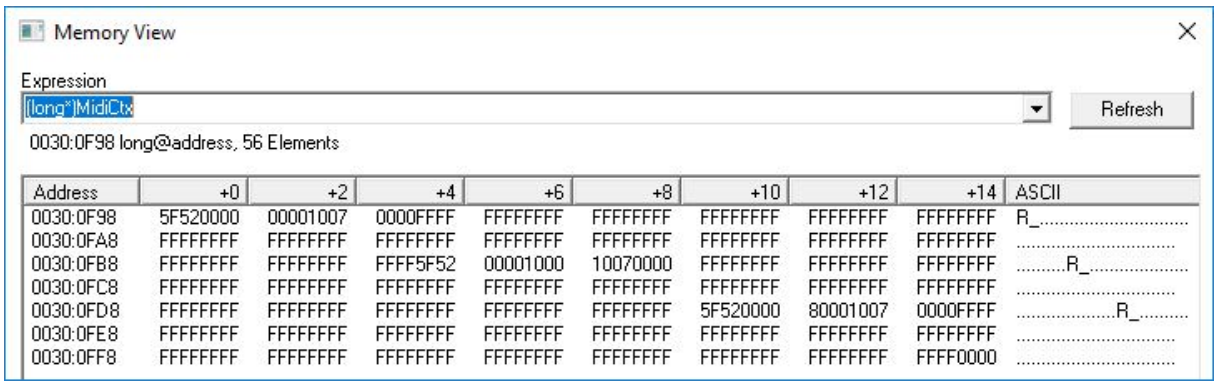
There only is a single field left to enter an expression and the output is only defined from this field. Since the MemoryView takes the same expression format as the watches, they may simply copy&pasted as required. In contrast to the watches the MemoryView supports three different display modes, which are automatically selected from the resulting expression type.

In this case the expression refers to an array of structures of type MIDICTX at memory address 0030:0F98. Since this is an array of structures, the chosen display format is a table.

If the expression refers to a single structure, display view changes into a list. We simply extend the expression with an array operator to see the second entry:



If your expression does not result in a structure, the output switches to the classic hexdump. To generate such a dump from the array of structures, simply typecast it to some scalar type, like WORD or long. The number of displayed elements is automatically determined from the underlying object:



Of course you may append a format specifier and override the quantity of elements.

7 Expression Parser

Watches, MemoryView and the hover display for both C and pure assembler projects now rely on the same, refurbished expression parser which, in turn, is aware of the complete debug information generated during the build phase.

For this reason all these instances now behave same and use the same expression syntax which is a subset of C expressions. The essential expression grammar looks like this:

```
displayexpression     : expression
                      : expression # optcount optfmt

expression      : IDENTIFIER
                | INTEGER
                | INTEGER : INTEGER
                | P16REG
                | PREFIX_PORT INTEGER
                | PREFIX_IRAM INTEGER
                | PREFIX_P24 INTEGER
                | PREFIX_P24 IDENTIFIER
                | expression '+' expression
                | expression '-' expression
                | expression '*' expression
                | expression '/' expression
                | expression '%' expression
                | expression '&' expression
                | expression '|' expression
                | expression '^' expression
                | expression '>>' expression
                | expression '<<' expression
                | '*' expression
                | '-' expression
                | '~' expression
                | '(' expression ')'
                | expression '.' IDENTIFIER
                | expression '->' IDENTIFIER
                | expression '[' expression ']'
                | '(' typespec ')' expression

typespec        : TYPENAME
                | typespec  '*'
                | typespec  'far' '*'
                | typespec  'const'

optcount        :

                | INTEGER

optfmt          :

                | DISPLAYFORMAT
```

The result of such an expression always is a reference to a memory location, where “memory” may be an address on async bus, a port, a location in IRAM or a P16 register. Along with that memory reference we get an exact type specifier which allows abstract operations with the data found on that location.

Valid expressions range from simple Symbol names to more advanced expressions with typecasts like “(long*)I:18h#4”, which effectively displays content of DR4..DR7.

Since expression results are always interpreted as memory references, typecasting is more graceful compared with C. For example if a structure member is referenced using an expression like

```
pStruct->EmbeddedStruct.SomeWordMember
```

Adding #8 will display 8 elements from SomeWordMember using its type.

7.1 Lexical elements

This section briefly describes the available lexical elements from the expression grammar.

- IDENTIFIER
  A potential identifier is any string starting with a letter character and only contains letters, digits or the underscore character. It only evaluates as an IDENTIFIER if it does not match a reserved word like const or one of the well-known types like WORD or long.
- INTEGER
  A decimal integer is any sequence of decimal digits (0..9). However, the lexer recognizes hexadecimal and binary values as well. Hexadecimal literals need to be prefixed with “0x” or end with a trailing “h”, like 0xABCD or 0FEDCh (please obey the required leading decimal digit in postfix notation), binary literals use prefix “0b” resp. a trailing “b”..
- P16REG
  P16 registers use their regular names like R3, XSP or DR7, but letters need to be uppercase.
- PREFIX_PORT
  The port prefix is “P:”, the following INTEGER is the port number.
- PREFIX_IRAM
  The IRAM prefix is “I:”, the following INTEGER is the IRAM address
- PREFIX_P24
  The P24 prefix starts with a single quote and an number between 48..63 followed by a colon, like “53:”, the following INTEGER is interpreted as an address inside that P24 memory.
- TYPENAME
  A TYPENAME is one of these case sensitive identifiers: “unsigned” or “WORD”, “signed” or “short”, “long” and “DWORD”.
- DISPLAYFORMAT
  The display format character defines the requested formatting for that item, its one of these:
  x: hexadecimal
  d or s: signed decimal
  u: unsigned decimal
  f: floatingpoint

dream

Contact : info@dream.fr

DREAM S.A.S.

Z.I., 21140 Semur-en-Auxois, France

www.dream.fr