Model Testing and Debugging on Real Hardware

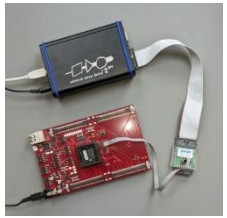
Jens Braunes

pls Development Tools

UDE® Universal Debug Engine

Complete tool set for debugging and trace

- On real target hardware
  - Microcontrollers, embedded systems, multicore SoCs



- On virtual prototypes / Simulators

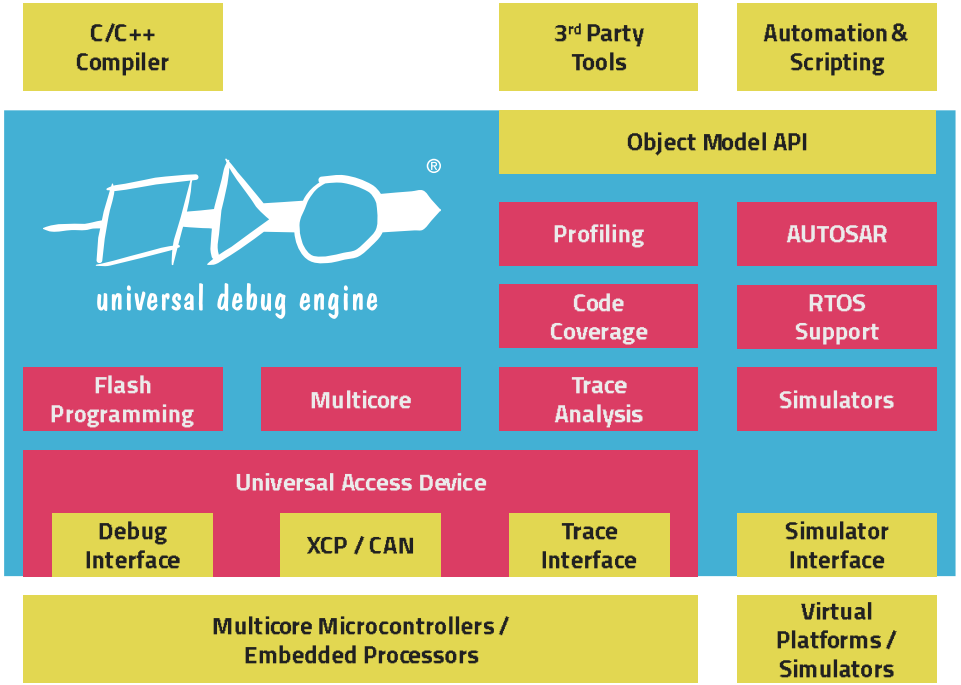
SYNOPSYS® COSEDA Technologies

- Real multicore debugging
  - System centric debugger environment, not core centric
  - One common user interface
  - Integrated Core Debuggers instead of separate debugger UI instances for cores

pls Development Tools

UDE® Universal Debug Engine

- Stop-mode multi-core debugging
  - Debug synchronization for multiple (heterogeneous) cores
  - Multi-core breakpoints
  - Multi-core / multi-program loading
- Trace based debugging
- Trace based run-time analysis
  - Profiling
  - Call graph analysis
  - Visualization of program execution
  - Performance analysis
  - Code coverage
- RTOS / AUTOSAR support
- Debug / test automation
  - Scripting
  - 3rd party tool coupling

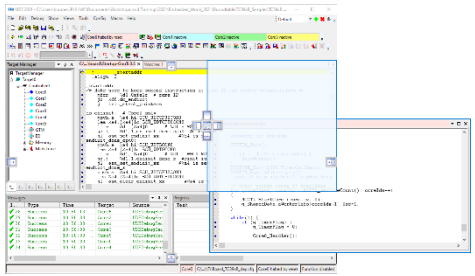


UDE® – Intuitive Tool for AURIX™ Multicore Debugging

Complete

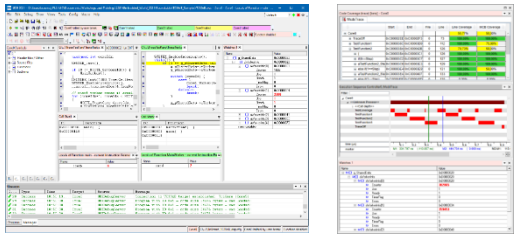
- Debugger environment for system, not for core
- All cores (main cores, special cores) visible
- One common user interface
- No separate debugger instances
- Core coloring

- Modern and state-of-the-art Window application
- Fully flexible
  - Arrange and group windows as needed



Multi-Screen Support

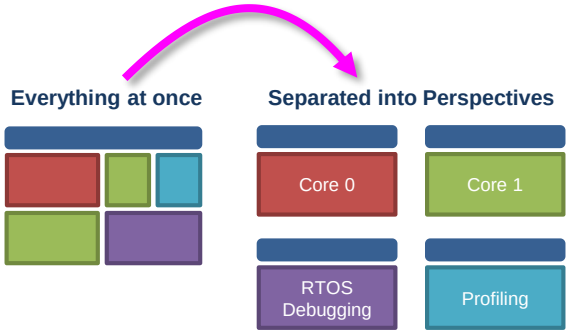
- Individual windows can be detached from main UDE window
- Detached windows can be grouped into docking containers



- Individual windows and dock containers can be moved to secondary screens

Perspectives

- Help to focus on the essentials
- One debugger session → multiple debug and test tasks → Perspectives
- Add perspectives as you like
- Freely configurable
  - In Layout
  - Contained windows (not only specific to one Core)



pls Development Tools

UDE Debug Support for AURIX™ TC3xx / TC4x

Main Cores

- Architecture
  - TriCore TC1.6, TC1.8
- Supported Languages
  - C, C++, Rust, Assembler
- Run-Control
  - HW and SW Breakpoints
  - Single stepping
  - Synchronization via UDE Run-Control Group
- Trace
  - Program, Data, Watchpoints, etc.
  - Parallel trace of cores (as selected)
  - TC3xx: subset of cores
  - TC4x: all cores

Generic Timer Module (GTM)

- Architecture
  - Bosch GTM 3.x, GTM 4.x
- Supported Languages
  - C, Assembler
- Run-Control
  - GTM 3.x (TC3xx): None (only debugging by tracing)
  - GTM 4.1 (TC4x): HW Breakpoints
  - Synchronization via UDE Run-Control Group
- Trace
  - MCS code trace
  - Data trace
  - Trace of signals

Parallel Processing Unit (PPU)

- Architecture
  - Synopsys ARC EV
- Supported Languages
  - C, C++, Assembler
- Run-Control
  - HW and SW Breakpoints
  - Single stepping
  - Synchronization via UDE Run-Control Group
- Trace
  - Program trace

Converter Digital Signal Processor (CDSP)

- Architecture
  - Synopsys ARC EM
- Supported Languages
  - C, C++, Assembler
- Run-Control
  - HW and SW Breakpoints
  - Single stepping

Cyber Security Real-Time Module (CSRM)

- Architecture
  - TriCore TC1.8
- Supported Languages
  - C, C++, Assembler
- Run-Control
  - HW and SW Breakpoints
  - Single stepping
  - Synchronization via Run-Control Group

Hardware Security Module (HSM)

- Architecture
  - Arm Cortex-M
- Supported Languages
  - C, C++, Assembler
- Run-Control
  - HW and SW Breakpoints
  - Single stepping
  - Synchronization via Run-Control Group

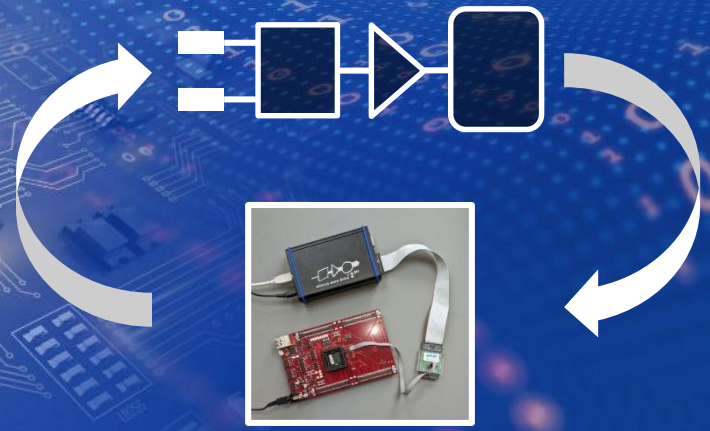
Stand-By Controller (SCR)

- Architecture
  - XC800 (8-bit)
- Supported Languages
  - C, Assembler
- Run-Control
  - HW and SW Breakpoints
  - Single stepping
- Trace
  - Not supported by HW
- Debug Interface
  - Separate SCR DAP I/F connected to some P33 port pins
  - Main SoC DAP port (also used for AURIX debugging)

TC2xx/3xx TC4x

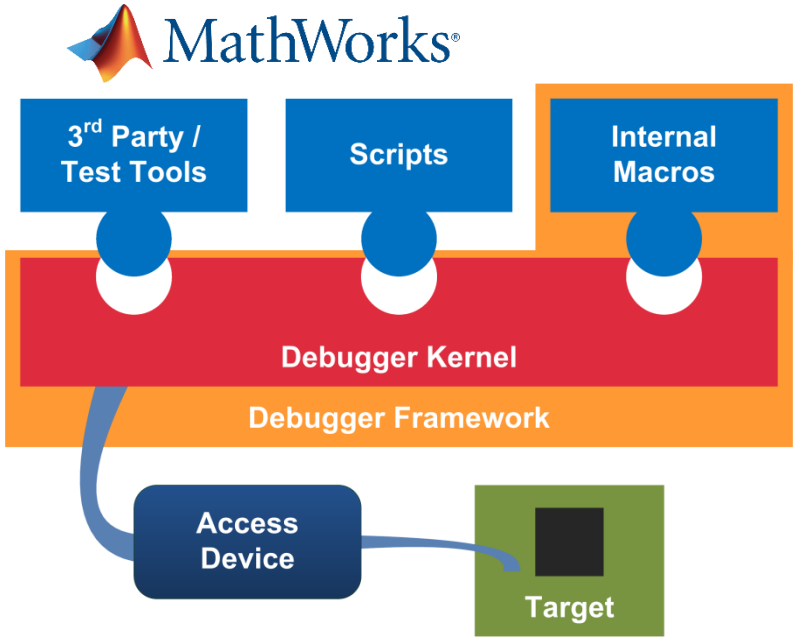
pls Development Tools

UDE Object Model for PIL/HIL Testing



UDE Automation by UDE Object Model

- Control the tool by other software
  - External scripts
  - Internal scripts (and macros)
  - 3rd party tools (e.g. **MATLAB**)
- Part of each installation → No additional license fee

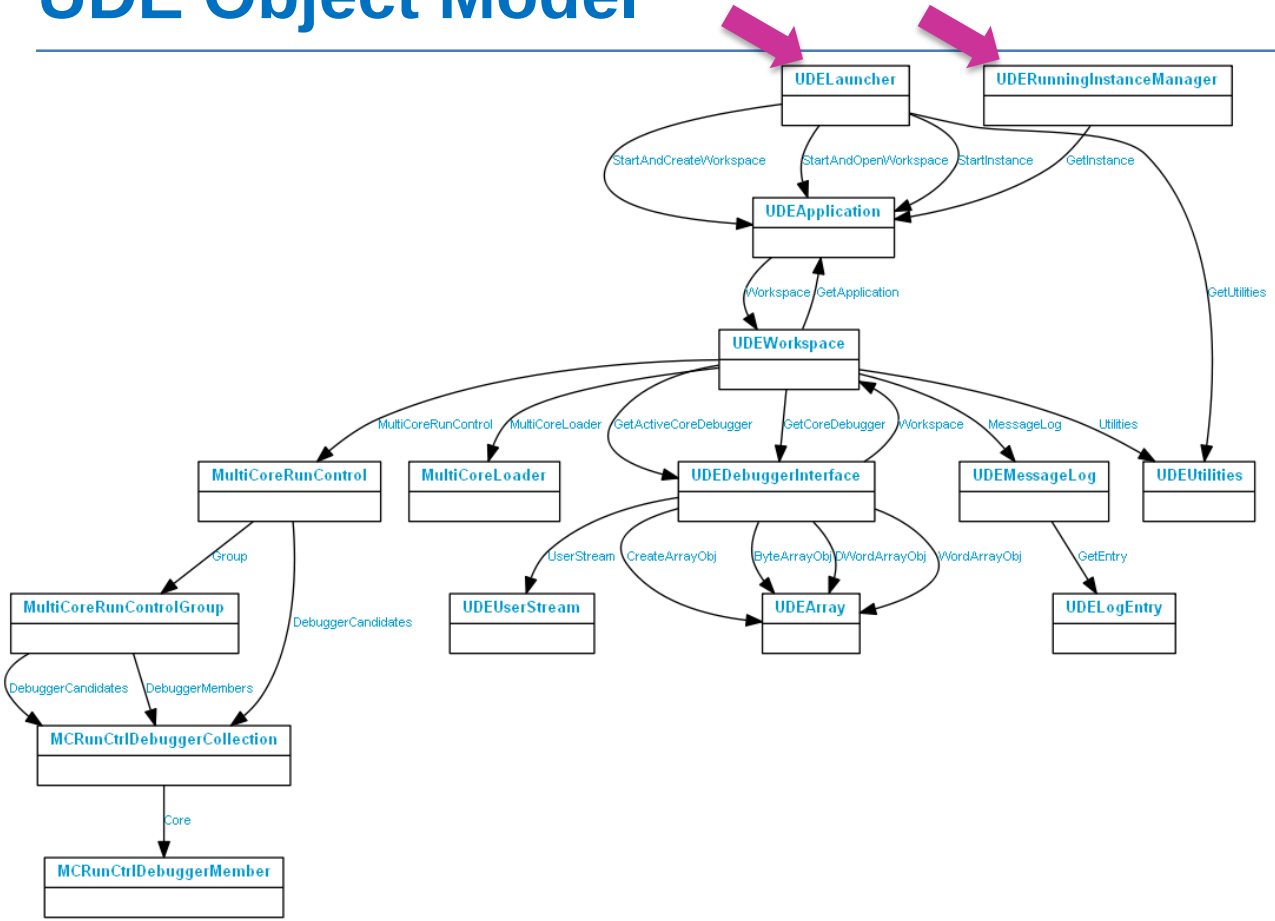


UDE Object Model

- Open and flexible software API based on Microsoft COM (**C**omponent **O**bject **M**odel)
- Access to all debugger functions

- No proprietary script language
- Support of common script languages which can use COM
  - JavaScript, **MATLAB**, Python, Perl, Windows PowerShell, etc.

UDE Object Model

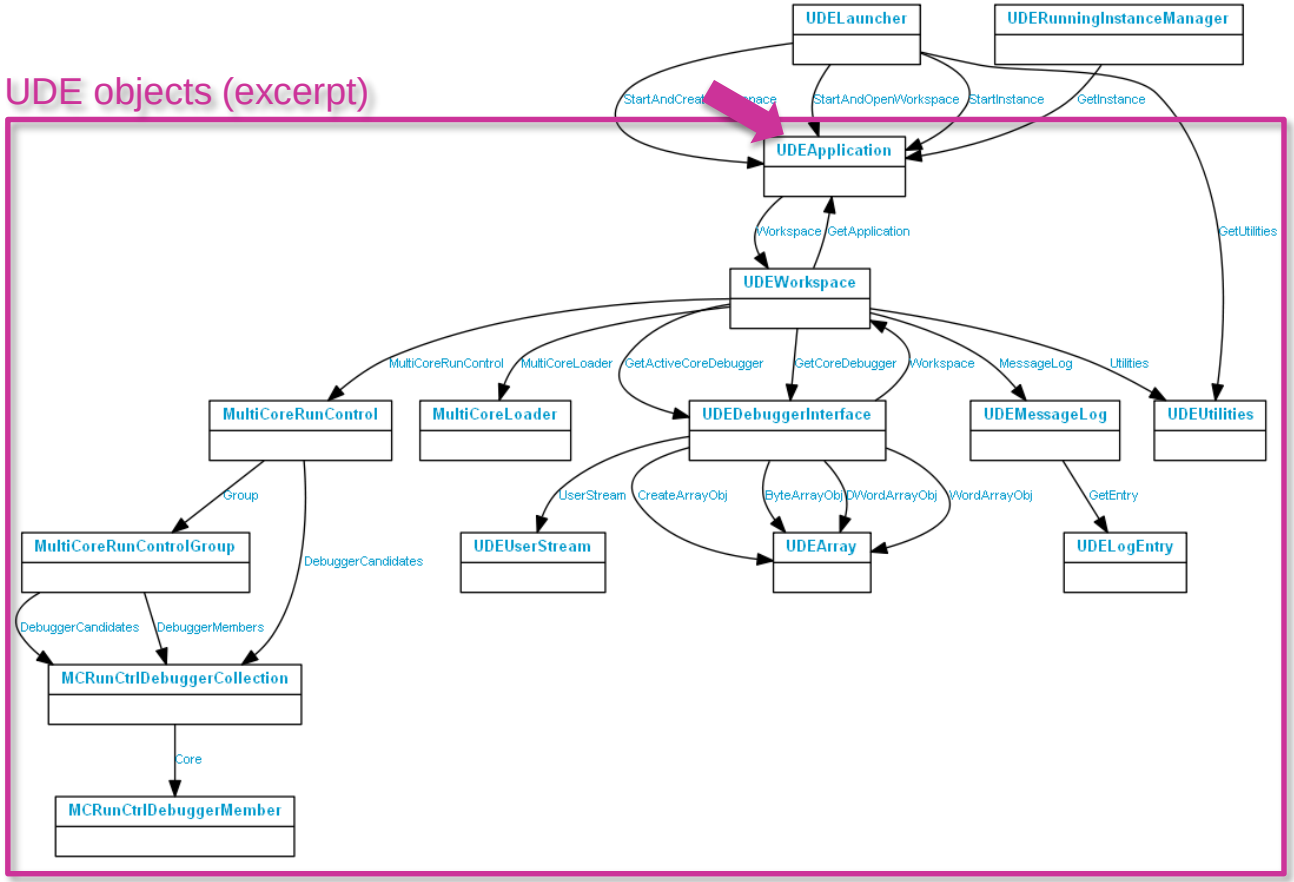


- UDE objects hierarchically accessible
- Two top level objects for scripts:
  - **UDE Launcher** for creating a new debugger instance
  - **UDE Running Instance Manager** for connecting an existing debugger instance

pls Development Tools

UDE Object Model

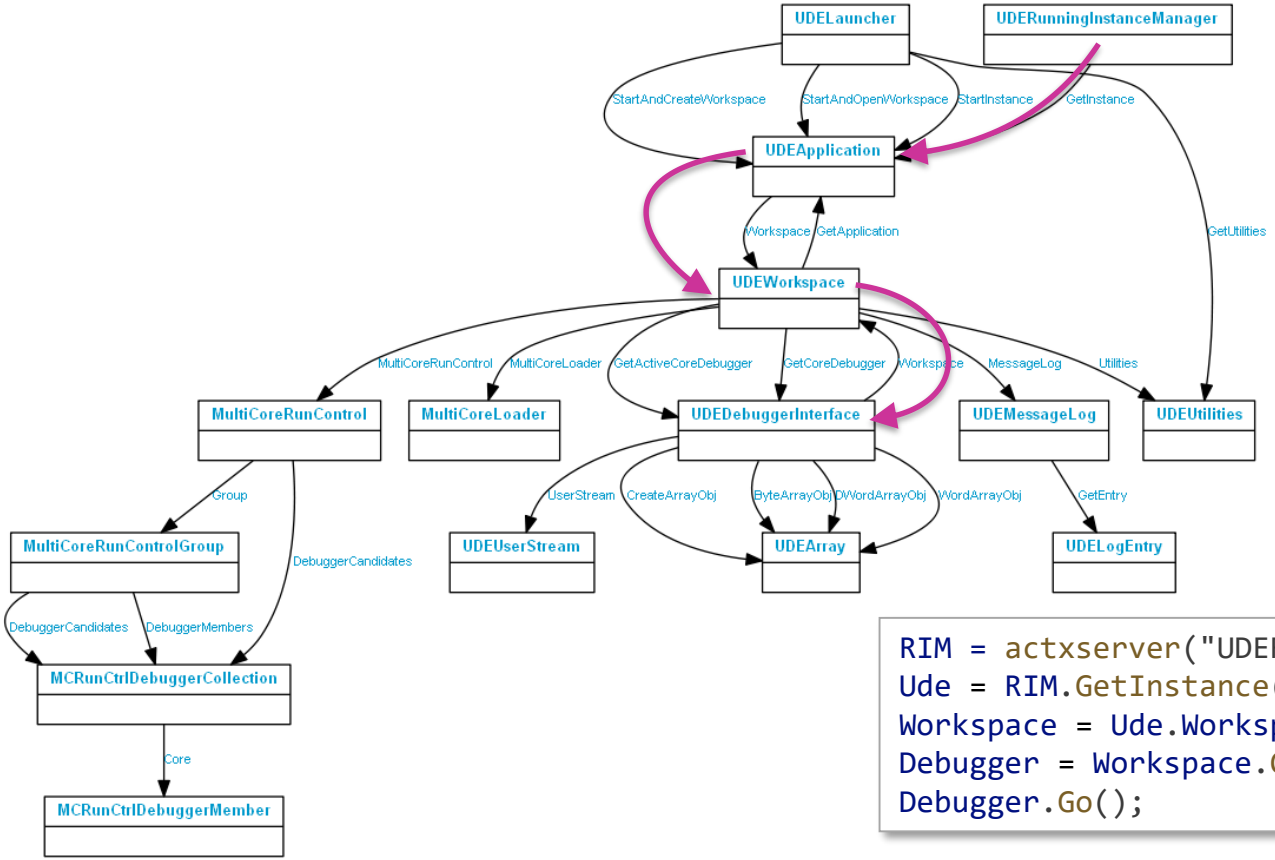
UDE objects (excerpt)



- **UDE Application** is the actual debugger instance
  - Hierarchical access to all debugger objects and functions from here

pls Development Tools

UDE Object Model



- **UDE Application** is the actual debugger instance
  - Hierarchical access to all debugger objects and functions from here
- Obtain child objects using
  - Get methods
  - Object properties

```
RIM = actxserver("UDERunningInstanceManager");
Ude = RIM.GetInstance(0);
Workspace = Ude.Workspace;
Debugger = Workspace.CoreDebugger(0);
Debugger.Go();
```


pls Development Tools

UDE Automation from MATLAB Script

- Create / start a new debugger instance
- Control the target application
- Set and get application state by writing / reading application values

Get UDELauncher

Start UDE and create new workspace with given target configuration file

Establish debugger connection to target device

```
%############## Start new UDE instance and create Workspace  #############%
ProgId = "UDELauncher";
Launcher = actxserver(ProgId);

% get current directory
WspFile = "D:\UDE_Wsp\Matlab_Sample\TimeDemo.wsx";
CfgFile = "D:\UDE_Wsp\Matlab_Sample\TriBoard_TC39xB_dap.cfg";
ElfFile = "D:\UDE_Wsp\Matlab_Sample\TimeDemo\IntRam\ClockDemo.elf";

% Start UDE and create a new Workspace
Ude = Launcher.StartAndCreateWorkspace("ude.exe", WspFile, CfgFile, 1);
Workspace = Ude.Workspace;

if ~Workspace.ConnectTarget(10000) % 10s timeout
    disp("ERROR: target not connected !");
    return;
end
```

Get core debugger object for Core0

Load ELF file into Flash if not already done

Start target application

```
% Get core debugger for Core0
Debugger = Workspace.CoreDebugger("Core0");

% 1st: check if elf file was already loaded
if ~Debugger.LoadAndFlash(ElfFile,"ProgramAndVerify")
    % 2nd: load and FLASH-program elf file
    if ~Debugger.LoadAndFlash(ElfFile)
        disp("ERROR: failed to load elf file !");
        return;
    end
End

Debugger.Go();
if ~Workspace.WaitForAllCoresRunning(1000, 0) % 1s timeout
  disp("ERROR: target not running !");
  return;
end
```

Add a breakpoint at function *UpdateSeconds*

```
% Set Breakpoint and wait for breakpoint hit
Debugger.Breakpoints.Add("UpdateSeconds");
if ~Debugger.WaitForHalt(3000)  % 3s timeout
  disp("ERROR: not halted at breakpoint !");
  return;
end
```

Write variables

```
% Write variables
Debugger.WriteVariable ("g_SharedData.Seconds", 59);
Debugger.WriteVariable ("g_SharedData.Minutes", 10);

Debugger.Go();
```


Read variables

```
% Read variables from target
Hours = Debugger.ReadVariable("g_SharedData.Hours");
Minutes = Debugger.ReadVariable("g_SharedData.Minutes");
Seconds = Debugger.ReadVariable("g_SharedData.Seconds");
disp("  Time: " + Hours + ":" + Minutes + ":" + Seconds);
```

Clean up

```
Debugger = NaN;
Workspace = NaN;

Launcher.StopInstance(Ude);
```

Documentation

IUDEDebuggerInterface Interface Reference

IUDEDebuggerInterface is the dual interface of debugger object model root object, which contains all basic debugger functions and functions to create or distribute child objects of the UDE object model. More...

Public Member Functions

HRESULT	Break (void)
HRESULT	ConInput ([out, retval] BSTR *pUserInput)
HRESULT	ConOutput ([in] BSTR UserOutput)
HRESULT	Expressions ([out, retval] LPDISPATCH *pExpressions)
HRESULT	GetFunctionReturnLocation ([in] VARIANT Location, [in, optional] VARIANT bSearchMode, [out, retval] LPDISPATCH *ppReturnInfoObject)
HRESULT	GetMemtool ([out, retval] LPDISPATCH *pVal)
HRESULT	Go (void)
HRESULT	LoadAndFlash ([in] BSTR Path, [in, optional] VARIANT Options, [out, retval] VARIANT_BOOL *pVal)
HRESULT	LoadBinDataFile ([in] BSTR Path, [in] VARIANT LoadAddr, [in] VARIANT_BOOL ProgFlash, [in, optional] VARIANT Options, [out, retval] VARIANT_BOOL *pVal)
HRESULT	LoadHexDataFile ([in] BSTR Path, [in] VARIANT_BOOL ProgFlash, [in, optional] VARIANT Options, [out, retval] VARIANT_BOOL *pVal)
HRESULT	LoadProgramFile ([in] BSTR pszProgramFile, [out, retval] VARIANT_BOOL *pbRetVal)
HRESULT	Read ([in] DWORD Adr, [in] VARIANT Value)
HRESULT	ReadMemory16 ([in] VARIANT Addr, [out, retval] VARIANT *pVal)
HRESULT	ReadMemory32 ([in] VARIANT Addr, [out, retval] VARIANT *pVal)
HRESULT	ReadMemory8 ([in] VARIANT Addr, [out, retval] VARIANT *pVal)
HRESULT	ReadRegister ([in] BSTR Register, [out, retval] VARIANT *pVal)
HRESULT	ReadVariable ([in] BSTR Expr, [out, retval] VARIANT *pVal)
HRESULT	ReloadProgram (void)
HRESULT	ResetTarget (void)
HRESULT	SetConnectState ([in] VARIANT_BOOL bVal)
HRESULT	SetLocation ([in] BSTR Description, [out, retval] VARIANT_BOOL
HRESULT	StepIn (void)
HRESULT	StepOut (void)
HRESULT	StepOver (void)
HRESULT	WaitForHalt ([in] LONG MaxWaitTime,[out, retval] VARIANT_BOOL
HRESULT	Write ([in] DWORD Adr, [in] VARIANT Value)
HRESULT	WriteMemory16 ([in] VARIANT Addr, [in] VARIANT NewVal, [out, retval] VARIANT_BOOL *pVal)
HRESULT	WriteMemory32 ([in] VARIANT Addr, [in] VARIANT NewVal, [out, retval] VARIANT_BOOL *pVal)
HRESULT	WriteMemory8 ([in] VARIANT Addr, [in] VARIANT NewVal, [out, retval] VARIANT_BOOL *pVal)
HRESULT	WriteRegister ([in] BSTR Register, [in] VARIANT NewVal, [out, retval] VARIANT_BOOL *pVal)
HRESULT	WriteVariable ([in] BSTR Expr, [in] VARIANT NewVal, [out, retval] VARIANT_BOOL *pVal)

Properties

LPDISPATCH	AddIns [get]
BSTR	BreakEvent [get]
LPDISPATCH	Breakpoints [get]
LPDISPATCH	ByteArrayObj ([in] long lSize) [get]
LPDISPATCH	CallStack [get]
VARIANT_BOOL	CanAccessWhileRunning [get]
VARIANT_BOOL	Connected [get]
LPDISPATCH	CoreInfo [get]
LPDISPATCH	DebuggerUserProfile [get]
LPDISPATCH	DisASMObj [get]
LPDISPATCH	DWordArrayObj ([in] long lSize) [get]
LPDISPATCH	Expression ([in] BSTR bsExpression) [get]
BSTR	Location [get, set]
VARIANT	Memory16 ([in] DWORD Adr) [get, set]
VARIANT	Memory32 ([in] DWORD Adr) [get, set]
VARIANT	Memory8 ([in] DWORD Adr) [get, set]
BSTR	Name [get]
LPDISPATCH	ProfilingSources [get]
BSTR	ProgramFile [get]
VARIANT	Register ([in] BSTR Register) [get, set]
	RTOSSupport ([in] BSTR pszAddInName) [get]
	State [get]
	Symbols [get]
	TargIntf [get]
	UserProfile [get]
LPDISPATCH	UserStream [get]
VARIANT	Variable ([in] BSTR Expr) [get, set]
LPDISPATCH	ViewsCollection [get]
LPDISPATCH	WordArrayObj ([in] long lSize) [get]
LPDISPATCH	Workspace [get]

Properties
- Get or set something

Methods:
- Actions for doing something

pls Development Tools

Documentation

Method

UDE internal definition, can be used by C++

```
ReadVariable ( [in] BSTR          Expr,
               [out,retval] VARIANT * pVal
             )
```

Get the value of a description, which describes the content or the location of a data variable of the current loaded program. This function is similar to property IUDEDebuggerInterface.Variable but easier to use by some programming languages.

Parameters

`[in]` **Expr** (`BSTR`) String variable that holds location description.

`[out]` **pVal** (`VARIANT*`) Pointer to return current value of variable.

Usage

C++

```
HRESULT hResult = UDEDebuggerInterface->ReadVariable(Expr, &Val);
```

C#

```
var Val = UDEDebuggerInterface.ReadVariable(Expr);
```

JScript

```
var Val = UDEDebuggerInterface.ReadVariable(Expr);
```

Perl

```
my $Val = $UDEDebuggerInterface->ReadVariable($Expr);
```

PowerShell

```
$Val = $UDEDebuggerInterface.ReadVariable($Expr)
```

Python

```
Val = UDEDebuggerInterface.ReadVariable(Expr)
```

VBScript / VisualBasic

```
Val = UDEDebuggerInterface.ReadVariable(Expr)
```

`UDEDebuggerInterface` is a variable that represents a IUDEDebuggerInterface object.

Code examples for different script languages

Property

Breakpoints get

Get breakpoint collection object - the object can be used to obtain currently used breakpoints or add and remove breakpoint objects. Function returns object of type IUDEBreakpoints.

Parameters

`[out]` **pBreakpoints** (`LPDISPATCH*`) Pointer to dispatch p… enumeration object .

Next Object / Interface of object hierarchy

Usage

C++

```
HRESULT hResult = UDEDebuggerInterface->Breakpoints
       (&Breakpoints);  // get
```

C#

```
var Breakpoints = UDEDebuggerInterface.Breakpoints;  // get
```

JScript

```
var Breakpoints = UDEDebuggerInterface.Breakpoints;  // get
```

Perl

```
my $Breakpoints = $UDEDebuggerInterface->GetProperty("Breakpoints");
       # get
```

PowerShell

```
$Breakpoints = $UDEDebuggerInterface.Breakpoints  # get
```

Python

```
Breakpoints = UDEDebuggerInterface.Breakpoints  # get
```

VBScript / VisualBasic

```
set Breakpoints = UDEDebuggerInterface.Breakpoints  ' get
```

`UDEDebuggerInterface` is a variable that represents a IUDEDebuggerInterface object.

pls Development Tools

Debug ■ Trace ■ Test

pls
Development Tools

www.pls-mc.com